



LUNDS UNIVERSITET  
Ekonomihögskolan

# Mjukvarutest i agil systemutveckling

En undersökning på ett multinationellt företag

Institutionen för informatik

Kandidatuppsats, 15 högskolepoäng

Framlagd: December, 2011

Författare: Viktor Ankarberg och Aron Tendler

Handledare: Hans Lundin

Examinatorer: Claus Persson och Andreas Jacobsson

**Titel:** Mjukvarutest i agil systemutveckling  
**Författare:** Viktor Ankarberg och Aron Tandler  
**Utgivare:** Institutionen för informatik  
**Handledare:** Hans Lundin  
**Examinatorer:** Claus Persson och Andreas Jacobsson  
**Publiceringsår:** 2012  
**Uppsattstyp:** Kandidatuppsats  
**Språk:** Svenska  
**Nyckelord:** Agil utveckling, vattenfallsutveckling, mjukvarutest, mjukvaruutveckling, testmetoder

### **Abstrakt**

Testning är en betydelsefull del i systemutvecklingsprocessen. Det är denna process som gör att kvaliteten garanteras i slutprodukten. Då systemutvecklingen på senare tid gått mot mer agila arbetsmetoder, så har vi undersökt hur testningen fungerar i en agil miljö. Vi har gjort vår undersökning på ett stort multinationellt företag som under de senaste åren genomgått en förändring av sin utvecklingsprocess. Företaget har gått från att utveckla enligt vattenfallsmodellen till att jobba med agila utvecklingsmetoder. Vi har kategoriserat testmetoderna som företaget använder sig av och undersökt hur de olika kategorierna används i agil utveckling. Vidare har vi jämfört detta med hur de olika testkategorierna användes i vattenfallsmodellen. Undersökningen visar att testmetoderna i sig inte har förändrats eller påverkats av övergången till det agila utvecklings sättet. Det som har förändrats är när testningen utförs, istället för att testa i slutet av utvecklingsprocessen testas det kontinuerligt, utvecklare och testare arbetat ihop. En annan förändring är att fokus ligger på strukturell testning i den agila utvecklingsmiljön.

# Innehållsförteckning

<b>1</b>	<b>Inledning.....</b>	<b>5</b>
1.1	Bakgrund .....	5
1.2	Problemområde.....	6
1.3	Frågeställning.....	6
1.4	Syfte.....	6
1.5	Avgränsning .....	6
<b>2</b>	<b>Teori .....</b>	<b>7</b>
2.1	<b>Utvecklingsmodeller.....</b>	<b>7</b>
2.1.1	Vattenfallsmodellen .....	7
2.1.2	Agil utveckling.....	8
2.1.3	SCRUM .....	9
2.1.4	Extremprogrammering.....	9
2.1.5	Testdriven utveckling.....	10
2.1.6	Testning i agil utveckling .....	11
2.2	<b>Testning och testmetoder .....</b>	<b>12</b>
2.2.1	Mjukvarutestning.....	12
2.2.2	ISO-9126.....	12
2.3	<b>Klassificering av testmetoderna.....</b>	<b>14</b>
2.3.1	Strukturell kontra funktionell systemtestning .....	14
2.3.2	Dynamisk kontra statisk systemtestning .....	15
2.3.3	Manuell kontra automatisk testning.....	15
2.3.4	Mobil kontra stationär testning .....	15
2.4	<b>Testmetoder .....</b>	<b>16</b>
2.4.1	Stress testing.....	16
2.4.2	Recovery testing.....	16
2.4.3	Security testing.....	16
2.4.4	Requirements testing.....	16
2.4.5	Regression testing .....	17
2.4.6	Error handling testing.....	17
2.4.7	Heuristic testing eller exploratory testing.....	17
2.5	<b>Ramverk .....</b>	<b>18</b>
<b>3</b>	<b>Metod .....</b>	<b>19</b>
3.1	<b>Undersökningsstrategi .....</b>	<b>19</b>
3.2	<b>Undersökningshjälpmedel .....</b>	<b>19</b>
3.2.1	Intervjuguiderna .....	19
3.3	<b>Datainsamling .....</b>	<b>21</b>
3.4	<b>Analys av material .....</b>	<b>21</b>
3.5	<b>Etik .....</b>	<b>22</b>
<b>4</b>	<b>Resultat .....</b>	<b>23</b>
4.1	<b>Den gamla utvecklingsmodellen .....</b>	<b>23</b>
4.2	<b>Testning i gamla utvecklingsmodellen.....</b>	<b>23</b>
4.3	<b>Den nya utvecklingsmodellen .....</b>	<b>24</b>
4.4	<b>Testning i den nya utvecklingsmodellen .....</b>	<b>25</b>
4.4.1	ISO-9126.....	26
4.4.2	Strukturell kontra funktionell testning.....	26
4.4.3	Dynamisk kontra statisk testning .....	27
4.4.4	Manuell kontra automatisk testning.....	27

4.4.5	Mobil kontra stationär .....	28
4.4.6	Testmetoderna .....	28
<b>5</b>	<b>Diskussion .....</b>	<b>30</b>
<b>6</b>	<b>Slutsats .....</b>	<b>32</b>
6.1	Förslag till fortsatt forskning.....	33
<b>Bilagor .....</b>	<b>.....</b>	<b>34</b>
	<b>Bilaga A - Intervjuguide 1 med beslutsfattarna .....</b>	<b>34</b>
	<b>Bilaga B - Intervjuguide 2 med beslutsfattarna.....</b>	<b>35</b>
	<b>Bilaga C - Intervjuguide testare.....</b>	<b>36</b>
	<b>Bilaga D - Anonymiserade transkriptioner.....</b>	<b>37</b>
<b>Referenser.....</b>	<b>.....</b>	<b>60</b>

# 1 Inledning

## 1.1 Bakgrund

Världen vi lever i blir mer och mer datoriserad. Mjukvara påverkar människor i stor utsträckning och många är idag vana vid att använda mjukvara i olika aspekter av sina liv, både privat och professionellt. Utvecklingen inom kommunikation och informationsteknologi gör att möjligheterna och efterfrågan på mjukvara ökar. Detta ställer höga krav på mjukvaruutveckling och utvecklarna att dels utveckla ny mjukvara men även att förbättra och anpassa den befintliga mjukvaran. Denna utveckling har gjort att mjukvaruutvecklingen har blivit en mer komplicerad process och projekten växer i omfattning. (Da Lucia et al, 2008)

Den första metoden som användes till mjukvaruutveckling var vattenfallsmodellen. Denna utvecklingsmodell används fortfarande till vissa mjukvaruutvecklingsprojekt. Här finns det en bestämd följd i hur utvecklarna skall utveckla mjukvaran och det finns inte någon möjlighet att backa tillbaka, utan det är ett flöde framåt likt ett vattenfall. I denna metod arbetar utvecklarna sig igenom de olika momenten i mjukvaruutvecklingen sekventiellt. (Avison och Fitzgerald, 2006)

Vattenfallsmodellen var tillräcklig i mjukvarans början, men för den större komplexitet som mjukvaruprojekten har idag är den för rigid och styrd (Da Lucia et al, 2008).

Mjukvaruutvecklingen har under de senaste åren genomgått en stor förändring. Agil utveckling är den nya trenden inom utveckling och det påverkar alla aspekter av hur mjukvaran tillverkas. Agil betyder smidig, vig och lättroblig, vilket antyder att agil utveckling eftersträvar att vara mer smidig än tidigare utvecklingsmodeller. I detta nya sätt att utveckla mjukvara är arbetssättet mer fritt och inte så bundet till olika faser. Metoden kom till eftersom många ansåg vattenfallsmodellen vara för styrd och otillräcklig. (Cockburn, 2002)

Mjukvaruutveckling är trots sin relativa ungdom ett område som ständigt förändras, nya metoder och teknologier växer fram kontinuerligt. En viktig del i mjukvaruutveckling är själva testningen och det är en nödvändighet att testa mjukvaran innan den är klar, för att säkerställa att mjukvaran fungerar tillfredsställande. Kvaliteten på mjukvaran är väldigt konkret och mätbar, man kan räkna hur många buggar och fel som finns. (Burnstein, 2003)

Mjukvarutestning är en del i utvecklingsprocessen samt en viktig kugge som säkerställer kvaliteten i den slutgiltiga mjukvaran. Utvecklingsmodeller inom organisationer förändras och anpassas till dagens rådande villkor (Burnstein, 2003). Testningen tar olika stor plats i olika utvecklingsmiljöer, antingen som en kontinuerlig process eller en slutstation när mjukvaran är färdig (Avison och Fitzgerald, 2006).

## 1.2 Problemområde

Det intressanta med testning är att det är den station där utvecklarna och testarna kan rätta till eventuella missar och problem i mjukvaran och säkra kvaliteten innan den slutligen hamnar i våra händer som konsumenterna.

Vi har gjort vår undersökning på ett stort globalt företag som hade bytt utvecklingsmodell från vattenfallsutveckling till agil utveckling. Det finns en hel del litteratur som beskriver testning av mjukvara, men inte så mycket som beskriver hur testningen skiljer sig mellan vattenfallsutveckling och agil utveckling i avseende testmetoder.

## 1.3 Frågeställning

När vi föresatte oss att undersöka detta ämne var det många frågor som vi undrade över men vi landade slutligen i följande två frågor:

- Hur fungerar mjukvarutest i agil systemutveckling på ett stort företag i avseende vilka testmetoder och testkategorier som används?
- Hur påverkar den organisatoriska strukturen av test- och utvecklingsorganisationen själva testningen av mjukvara?

## 1.4 Syfte

Vårt syfte med uppsatsen är att kartlägga hur mjukvarutestning fungerar i avseende vilka testmetoder och testkategorier som används. Detta jämför vi med hur det fungerade när företaget använde sig av vattenfallsmodellen.

## 1.5 Avgränsning

Vi har avgränsat uppsatsen till att undersöka hur själva utvecklingsmodellen påverkar testningen. Vi utesluter inte att det finns andra områden som kan påverka testningen men har valt att undersöka denna faktor. Vi har även valt att avgränsa undersökningen till hur denna påverkan ser ut på en stor organisation och utesluter inte att det är annorlunda i mindre organisationer. Fortsatt så undersöker vi inte de tekniska aspekterna av testens utfall eller resultatet av testningen utan enbart hur påverkan är på testmetoderna och valet av dessa. Vi gör ingen bedömning av vilket som är bättre eller sämre i förhållande till antal buggar och effektivitet utan enbart hurdan förändringen är.

## 2 Teori

### 2.1 Utvecklingsmodeller

Vi kommer här att presentera vattenfallsmodellen och tre stycken agila modeller som används på företaget där vi har gjort vår undersökning. De tre agila modellerna är Testdriven utveckling, SCRUM och Extremprogrammering.

#### 2.1.1 Vattenfallsmodellen

Vattenfallsmodellen är en beprövad och relativt gammal utvecklingsmodell. Det var ungefär 40 år sedan som den etablerades som en av de ledande mallarna för system och mjukvaruutveckling (Görling, 2009). Modellen består av fem steg:

- Kravanalys och specifikation
- Design
- Implementering
- Testning och verifikation
- Installation och underhåll

I det första steget, kravanalys och specifikation, så identifieras vilka krav som finns på produkten som ska utvecklas och sedan så skapas en specifikation utifrån detta. Ofta så handlar det om att titta på ett nuvarande system, dess problem, vad som kan göras åt detta och om det ens är möjligt. Steg två, design, använder sig av specifikationen från steg ett för att skapa en design av mjukvaran.

Nästa steg är implementation. Där sker själva programmeringen och den utgår från designen som skapades i föregående steg. Med andra ord så översätts designen till kod. Komponenterna i mjukvaran testas även här, men bara var för sig och inte som en helhet som det görs i nästa steg (Stober & Hansmann, 2010). När steg fyra, testning och verifikation påbörjas, så sätts alla komponenter ihop till ett system. Utvecklarna har nu uppgett att de är färdiga med implementeringsfasen och testningen kan påbörjas på allvar. Det är ofta här som projekt tenderar att falla i denna utvecklingsprocess. Mjukvaran ska i princip vara helt färdigställd så att alla de tester som behövs för att säkerställa de krav som ställdes i steg ett görs.

Steg fem innebär installation och underhåll. Det är nu som mjukvaran överlämnas till kunden. Mjukvaran underhålls dock fortsättningsvis och kontinuerligt. Support till kunder är väldigt vanligt (Görling, 2009).

Innan nästa steg påbörjas måste det föregående steget vara klart och där i itereras processen alltså i fasen som

utvecklingen befinner sig i finns möjlighet att börja om från början och ändra på saker, men det är enbart där. När nästa steg inleds så försvinner möjligheten att modifiera något i ett tidigare steg eftersom modellen är en sekventiell utvecklingsmodell ett steg i taget tas. När ett nytt steg i utvecklingen påbörjas så är det svårt att gå tillbaka och ändra något, vattenfallet rinner i en riktning och kan inte backa. Många företag går ifrån detta sätt att utveckla för att använda sig av en agil utvecklingsmodell men det finns fortfarande utveckling där denna modell lämpar sig bäst. (Görling, 2009)(Avison och Fitzgerald, 2006)

På grund av den ursprungliga vattenfallsmodellens brister så har det gjorts förändringar i denna i form av en modifierad vattenfallsmodell. Den stora skillnaden mellan den gamla och den modifierade vattenfallsmodellen är övergången mellan faserna i processen. I den modifierade så överlappas faserna, vilket betyder att det är alltid är två faser igång samtidigt. Detta ska göra så att hanteringen av plötsliga förändringar i utvecklingsprocessen blir enklare och smidigare. (Satalkar, 2010)

Trots förbättringarna i modellen så har även den modifierade vattenfallsmodellen sina brister. Flexibiliteten har ökat men på grund av att faserna nu mer överlappar varandra så har det uppstått svårigheter i att följa upp utvecklingsprocessens fortlöpande. (Satalkar, 2010)

### **2.1.2 Agil utveckling**

År 2001 samlades grundarna av den agila skolan för ett möte där ett manifest om vad som var viktigt inom agil utveckling deklarerades (Avison och Fitzgerald, 2006). Det som uppdragades var att individer och interaktioner värderas över processer och verktyg, fungerande mjukvara över omfattande dokumentation, samarbete med kunden över kontraktsförhandling samt att reagera på förändringar istället för att bara följa en plan (Beck et al, 2001).

Fenomenet agil utveckling myntades under 1990-talet men även om det inte är helt nytt och främmande så anser många det vara relativt svårt att precis definiera innebörden av ordet agil inom utveckling (Highsmith, 2002 citerad i Avison och Fitzgerald 2006). Agil utveckling är en samling iterativa utvecklingsmetoder där förändring i processen förespråkas för att uppnå högsta kvalitet (Beck et al, 2001). Det finns en hel del olika varianter av agila utvecklingsmodeller men de flesta härstammar från en traditionell utvecklingsmodell, t ex vattenfallsmodellen. Den agila skolan har en uppfattning om att användare av en mjukvara, eller egentligen en framtida användare av en mjukvara, ofta har svårt för att artikulera och definiera detaljerat vad deras krav på mjukvaran är. De är inte tillräckligt tydliga i många fall även om de själva tror det. Men när mjukvaran väl är färdig så visar det sig att det inte riktigt var vad de ville ha (Avison och Fitzgerald, 2006). Den stora skillnaden från tidigare sätt att arbeta var det mer flexibla förhållningssättet till arbetsmetoderna i agil utveckling. Samarbetet är närmare mellan utvecklaren, användaren eller beställaren som under hela utvecklingsprocessen skall konsulteras och rådgöras med. Testning och utveckling skiljs inte åt utan testningen blir en integrerad del av utvecklingen. Agil utveckling är ett samlingsnamn för ett antal olika utvecklingsmetoder, bland annat SCRUM, testdriven utveckling och XP (Extreme programming). (Cockburn, 2002)



### 2.1.3 SCRUM

SCRUM är en utvecklingsmetod som skiljer sig på flera sätt från exempelvis vattenfallsmodellen. Istället för att ha en sådan utpräglad steg-för-steg modell så utgår utvecklarna från att det är svårt att planera i förväg exakt hur det ska gå till i utvecklingsprocessen. Byråkratin minimeras, uppdelning av projektet mellan olika personer existerar inte på samma sätt och tidsrapportering existerar inte på samma sätt. Anpassning är ett av nyckelorden i den här utvecklingsprocessen just på grund av att det hela tiden strävas efter att göra det som är bäst för projektet just då. (Görling, 2009 och Avison och Fitzgerald, 2006)

Huvudsyftet med denna metod är att skala bort den typ av byråkrati och planering som egentligen inte är särskilt nödvändig för projektet. Utveckling är oförutsägbar enligt SCRUM. (Görling, 2009)

Utvecklingen sker i sprintar som är små iterationer på omkring två veckor. Dessa sprintar består av design, utveckling, testning och dokumentation. En sprint varar vanligtvis runt två veckor men det kan variera från projekt till projekt. (Stober & Hansmann, 2010)

Inom SCRUM finns det tre roller, tre dokument och tre möten. Rollerna är produktägare, team och SCRUM master. Produktägaren är den som har det övergripande ansvaret för den slutgiltiga produkten. Det är viktigt att produktägaren representerar alla intressenters intressen och definierar kraven. Han eller hon är ekonomiskt ansvarig och har sista ordet i projekten. Utvecklarna är teamet och de utvecklar och testar mjukvaran. SCRUM-mastern är projektledaren som har ansvar för "SCRUM-processen" och att anpassa SCRUM till projektet och organisationen. SCRUM-masterns viktigaste uppdrag är att säkerställa att teamet levererar hög kvalitet på mjukvaran och kunna diskutera detta med teamet och produktägaren. (Stober & Hansmann, 2010)

De tre dokumenten är "Product backlog", "sprint backlog" och "sprint result". "Product backlog" är en lista med krav på funktioner för produkten. "Sprint backlog" är en lista med prioriterade uppgifter som ska genomföras under sprinten och slutligen är "sprint result" en sammanställning av vilka uppgifter som löstes och vilka som kvarstår. (Stober & Hansmann, 2010)

De tre mötena är sprint-planeringsmöte, dagligt SCRUM-möte och sprintgenomgång. Varje sprint börjar med sprintplaneringsmöte där teamet, SCRUM-mastern och produktägaren bestämmer sig för en "sprint backlog" baserat på kraven prioriterade av produktägaren och en bedömning av vad teamet har gjort och därav kan förpliktiga sig att göra under sprinten. Det dagliga SCRUM-mötet är ett ca 15 minuter långt möte som går genom vad som har blivit uppnått de senaste 24 timmarna och eventuella problem diskuteras. I slutet av varje sprint träffas alla inblandade på en sprintgenomgång för att stämma av var projektet befinner sig efter sprinten. (Stober & Hansmann, 2010)

### 2.1.4 Extremprogrammering

Precis som SCRUM, så ingår extremprogrammering i agila utvecklingsmodeller. I denna utvecklingsmodell så är tanken att nya versioner av mjukvaran skapas väldigt ofta och att fungerande, enkel kod ska skapas så tidigt som möjligt. Detta för att man vill kunna börja testa den så fort som möjligt så att felen och buggarna hittas

tidigt och inte i slutskedet. (Görling, 2009)

Extremprogrammering tillhör den agila utvecklingsmodellen där förändringar uppmuntras i mjukvaran även i ett sent skede. Testningen sker helt parallellt med de andra aktiviteterna i utvecklingen under hela livscykeln och automatiska tester är något som förespråkas flitigt. (Görling, 2009)

I extremprogrammering så sitter ofta utvecklarna två och två. De sitter båda vid samma dator och skärm och turas om med att skriva kod. Den som inte programmerar sitter bredvid och ger kommentarer och förslag på förbättringar. Detta har visat sig vara ett effektivt sätt att utveckla på och denna utvecklingsmetod ska kunna fungera i både stora och små projekt (Eriksson, 2006). Projekt med 3-10 programmerare rekommenderas dock (Avison och Fitzgerald, 2006).

En annan viktig punkt i extremprogrammering är *refactoring* (omstrukturering av kod) där koden förändras och förbättras utan att andra komponenter i mjukvaran påverkas av detta. Exempel på hur detta går till är att kod tas bort som inte används längre och försöker göra koden mera lättläst för andra ögon än sina egna. Enkelhet är ett annat ord som betonas mycket. Utvecklarna ska fokusera på den funktionaliteten som behövs och är bestämd tidigare så att inte onödig kod skrivs. (Stober & Hansmann, 2010)

Kontinuerlig testning är även i denna utvecklingsmodell en mycket viktig punkt. Görs inte detta, så löper risken att ändringar som görs på ett område i koden förstör för ett annat. (Stober & Hansmann, 2010)

### **2.1.5 Testdriven utveckling**

Denna utvecklingsmetod påminner om Extreme Programming där koden är den vanligaste och viktigaste dokumentationsformen och fokus ligger på iteration och mycket testning. Det som utmärker denna utvecklingsmodell är att det finns en stor mängd med tester som gör det lättare att gå in och ändra i programkoden utan att förstöra. (Görling, 2009)

Automatisering är ett av nyckelorden när det kommer till testning i en så kallad testdriven utveckling. Eftersom tanken i denna utvecklingsmodell är att alla tester ska göras om väldigt många gånger, så krävs det alldeles för mycket arbetskraft och pengar om all testning skulle göras manuellt. (Stober & Hansmann, 2010)

I denna utvecklingsmodell så har utvecklaren även en stor del i testningen. En av hans arbetsuppgifter är att designa *testcase* som sedan används när det ska testas (Stober & Hansmann, 2010). I ett *testcase* så står det hur funktionen fungerar, hur testet bör gå till och vad som förväntas att hända när det utförs (Burnstein, 2003). När testningen är klar så börjar cykeln om och utvecklaren gör de ändringar i koden som behövs för att sedan skicka över det återigen för exakt samma testning. Detta går hela tiden runt tills testningen blir godkänd (Stober & Hansmann, 2010).

Hänsyn ska tas, som i många andra utvecklingsmodeller, beroende på hur stort projektet är och hur många människor som är inblandade. Vissa delar i utvecklingsmodellen passar bättre för stora projekt och andra för små. Till exempel så blir iterationerna i utvecklingen längre ju större projektet är. Det är även av olika skäl, som till exempel mindre omfattande kod, betydligt enklare att vara mer insatt i vad de andra utvecklarna gör ju mindre projektet är (Stober & Hansmann, 2010)

### 2.1.6 Testning i agil utveckling

Sättet att bedriva testning av mjukvara i organisationer kan variera beroende på vad för slags utvecklingsmodell som användes. De senaste tio åren har utvecklingsmetoder, mer eller mindre förändrats till att vara mer agila. Från att vara ett steg i processen, oftast i slutet som i exempelvis vattenfallsmodellen, så är testning ofta numera integrerat i hela utvecklingsprocessen. I utvecklingsmodeller som härstammar från vattenfallsmodellen så förläggs testningen i slutet av utvecklingens livscykel. Även om vissa aktiviteter bör läggas parallellt med varandra så fungerar det inte på det sättet i vattenfallsmodellen. Självklart måste ju testning mer eller mindre ske kontinuerligt även i vattenfallsmodellen, även om det bara handlar om att testa och se så att koden som nyss skrivits är korrekt (Eriksson, 2006).

Övergång från traditionella utvecklingsprocesser till agila innebär också andra förändringar i testningsprocessen (Crispin, 2006). Agila utvecklingsmodeller omdefinierar begreppet kvalitetssäkring och de agila principerna kräver att utvecklare och testare förenas och arbetar ihop. Den huvudsakliga skillnaden från traditionella testningsprocesser är att i agila förhållanden så testas inte bara testarna utan det gör även utvecklarna. Alla inblandade i utvecklingsprocessen har alltid varit ansvariga för att kvaliteten säkras men i agila utvecklingsprocesser så skriver även utvecklarna testcase (Dubinsky et al, 2006). En annan radikal förändring i processen är att utvecklare nu allt oftare faktiskt till och med komponerar testcase innan de ens har skrivit koden i mjukvaran. Detta gör att de måste att de måste tänka på många olika aspekter som rör den nya funktionaliteten innan kodningen påbörjas. (Crispin, 2006).

En studie gjord på Ericsson 2007 om övergången mellan vattenfallsutveckling till agil utveckling visar att den kontinuerliga testningen i agil utveckling är mer effektiv och antalet buggar är färre i den agila utvecklingen än i vattenfallsutveckling dels på grund av felet upptäcks tidigare i processen och man kan spåra vem som är ansvarig för felet, vilket ger en incitament att leverera bättre mjukvara men även så kan den ansvariga rätta till sina egna fel. En annan orsak till förbättringen är att de stora blocktesten i slutet som görs i vattenfallsutveckling är sämre lämpade än kontinuerliga test är att det blir mer komplicerat och svårare att rätta till fel i ett färdigt system som sitter ihop än enskilda funktioner på vägen. En tredje anledning som angavs i studien är att när testare och utvecklare sitter tillsammans som de gör i agil utveckling så är det lättare att kommunicera med varandra verbalt istället för skriftligt vilket ger en större förståelse för varandras uppgifter och mindre risk att missförstånd uppstår. (Petersen, K & Wohlin, C, 2010)

Det testas betydligt mer i agila utveckling än i t ex vattenfallsmodellen. Det är inte ovanligt att testning som täcker i stort sätt hela mjukvaran genomförs veckovis. Iteration och regression är två nyckelord. (Dubinsky et al, 2006)

Nu när även utvecklare hjälper till i testningsprocessen så kan det tyckas vara överflödigt att använda sig av professionella testare, men så är inte fallet. Testarna hjälper utvecklarna med att skriva en del av deras testfall, eftersom de är bättre på just detta. Det som är utmaningen när en organisation arbetar på detta sätt är att koordinera all testning mellan testare och utvecklare. (Dubinsky et al, 2006)

## 2.2 Testning och testmetoder

### 2.2.1 Mjukvarutestning

Datorer och mjukvaror spelar nuförtiden, och har gjort det senaste decenniet, en oerhört stor roll i våra liv. Många IT-företag och utvecklare konkurrerar om vår uppmärksamhet och det har gjort att de pressas hårt för att få en hög kvalitet i sina mjukvaror. Dålig kvalitet accepteras överhuvudtaget inte längre. Tidigare insåg inte utvecklare hur viktigt det är med testning och att man genom den ser till att det hålls hög kvalitet på sin mjukvara, så fanns det inte riktigt personal som var specialiserad på att just testa och hitta fel i mjukvara. Koden skrevs, undersöktes till att den var korrekt och sedan var det bra med det. Utvecklare man mjukvara idag så behövs det mer eller mindre specialiserade personer för testning. (Burnstein, 2003)

Systemutvecklare, eller mjukvaruutvecklare, har genom åren blivit duktiga på att utveckla metoder och sätt att förhindra så att fel i mjukvaran uppstår. Det är dock svårt att undvika och det är tveksamt om den perfekta mjukvaran existerar. Så, det viktiga är egentligen inte att undvika och eliminera felen under utvecklingen utan att de hittas innan de skickas ut till sina kunder. (Burnstein, 2003)

Om en utvecklingsprocess hade varit perfekt så hade testning av mjukvaran varit överflödigt. Utförs utvecklingen exakt som det ska och kravspecifikationen följs så är testning en tidsberövande process. Nu finns det inte någon perfekt utvecklingsprocess så testning är numera en självklarhet och kommer att förbli en väldigt viktig del i utvecklingsprocessen. Testning kan också ses som ett slags informationssamlade av testaren som programmeraren eller utvecklaren kan använda i framtiden för att undvika att göra misstag som tidigare gjorts. (Perry, 2000)

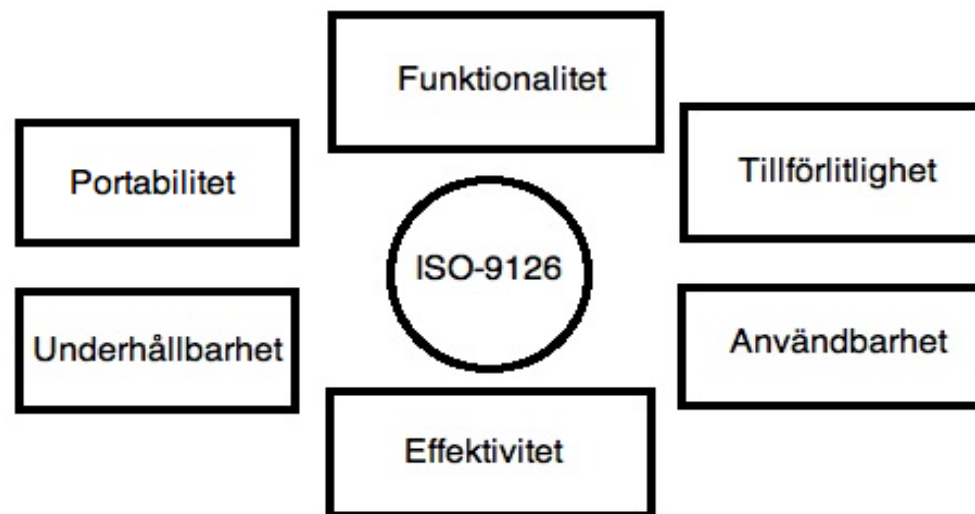
Perry (2000) säger att två tredjedelar av alla de fel som upptäcks kan härledas till några av de första faserna i utvecklingsprocessens livscykel vilket då betyder att de här felen och buggarna skrivs och kodas in i mjukvaran innan någon form av testning har utförts. Testning bör därför medräknas i alla faser i livscykeln och detta är någonting som syns mer och mer. Att bara ha en testfas i slutskedet av projektet håller inte.

Det finns många svårigheter med testning fortsätter Perry (2000). Tid och kostnader är t ex aspekter som så ofta ett problem. Både för lite och för mycket tid kan läggas på testningsprocessen och varken eller är något som är positivt. Det är en stor svårighet att anpassa testningsprocessen så att den är tillräcklig men inte tar för lång tid. ”Tid är pengar” är ett uttryck som passar in här. Det eftersträvas helt enkelt att hitta felen och buggarna så tidigt som möjligt så att kostnaderna hålls nere. Det gäller även att testarna och utvecklarna har en nära dialog och kommunikation så att det helt säkert att de felen som hittats även rättas till.

### 2.2.2 ISO-9126

ISO-9126 är namnet på en internationell standard som används vid utvärderande av kvaliteten på mjukvaran. Enligt denna standard, så är *kvalitet* definierat som ett antal olika drag eller kännetecken på en produkt (mjukvara i det här fallet) som ska tillfredsställa kraven och behoven från framförallt beställaren av produkten. ISO-9126 består av sex stycken betecknande som bör ses över för att säkerhetsställa kvaliteten på produkten:

- **Funktionalitet:** Produktens förmåga att tillhandahålla de funktioner som det är krav på.
- **Tillförlitlighet:** Produktens förmåga att bibehålla sin prestanda under de förhållanden som beställaren har krav på.
- **Användbarhet:** Detta handlar om hur enkelt det är att lära sig och förstå produkten eller hur stor kraftansträngning som behövs för att använda den.
- **Effektivitet:** Här undersöks hur effektiv produkten är i förhållande till hur mycket resurser som används och tidsaspekten.
- **Underhållbart:** Hur lätt är det att underhålla och modifiera produkten?
- **Portabilitet:** Portabilitet betyder i denna kontext hur lätt det är att flytta produkten från en miljö till en annan.



Figur 1: Överblick över ISO-9126

## 2.3 Klassificering av testmetoderna

Beroende på vad som ska testas och vad det är för resultat önskas uppnås så finns det enligt Perry (2000) en relativt stor mängd olika testmetoder för mjukvara. Val av testmetod kan vara en väldigt komplicerad process där en stor mängd aspekter måste tas hänsyn till (Perry, 2000).

Det finns många olika tekniska aspekter som iakttas men de som kanske är av mest avgörande är tidsaspekten och den ekonomiska aspekten. När allt kommer omkring så finns det alltid en deadline i utvecklingen och den ekonomiska faktorn kan också främja eller hämma. Ofta så är tidsschemat väldigt pressat och tiden är inte lång innan mjukvaran ska släppas på marknaden. Det gör att mjukvaran kan släppas med många buggar och fel. (Burnstein, 2003)

Perry (2000) beskriver att testmetoder kan delas upp i enlighet med deras egenskaper.

- Strukturell kontra funktionell testning
- Dynamisk kontra statisk testning
- Manuell kontra automatisk testning
- Mobil kontra stationär testning

Vi kommer nedan att gå igenom olika testmetoder och tekniker som ligger i de olika kategorierna.

### 2.3.1 Strukturell kontra funktionell systemtestning

Om en testare som speciellt är ute efter att hitta fel och buggar som uppstått under kodningen av mjukvaran, ska han eller hon utföra tester som härstammar från strukturell testning. Även om just testning av kod kan tyckas göras i början av ett projekt så är strukturella testningsmetoder något som kan köras under hela livscykeln. Strukturella tester ska hitta fel och buggar som förekommer under själva kodningen av mjukvaran och den ska garantera tillräcklig testning så att en funktion kan implementeras i mjukvaran. Syftet med denna sorts testningsmetoder är inte att se till så att funktionerna i mjukvaran fungerar som det ska utan det är helt enkelt strukturen. Är inte strukturen korrekt från början så är det meningslöst att ens börja lägga in funktioner. Några exempel på testmetoder som faller in under konceptet strukturell testning är ”stress testing”, ”recovery testing” samt ”security testing”. (Perry 2000)

Systemtester utförs på systemnivå för att försäkra sig om att systemet eller mjukvaran vidhåller de krav som finns i kravspecifikationen och betar sig som det är menat (Burnstein, 2003). Som namnet antyder så görs tester på systemets olika funktioner och fokus ligger på att de inputs och outputs som ska respektive inte ska gå att utföra. Exempel på testningsmetoder som, enligt Perry (2000), går under funktionell systemtestning är ”requirements testing”, ”regression testing” samt ”error handling testing”.

### **2.3.2 Dynamisk kontra statisk systemtestning**

Testmetoder och tekniker kan även klassificeras som antingen dynamiska eller statiska enligt Perry (2000). Enkla förklaringar av dessa två egenskaper är att under dynamisk testning så körs och exekveras mjukvaran eller applikationen men under statisk testning så brukar mjukvaran inte köras. Dynamisk testning kräver helt enkelt att mjukvaran körs och i analys av resultatet så undersöks det hur den uppförde sig under exekveringen. Under statisk testning behöver inte mjukvaran vara igång utan ett statistiskt test kan till exempel vara att mjukvarans kod och syntax studeras. (Perry, 2000)

Generellt sett så utförs statiska tester relativt tidigt i processen, under till exempel designfasen, och dynamiska tester arbetas det då oftare med när det är dags att testa mjukvarans funktioner och beteende. För att mjukvaran ska fungera korrekt under de dynamiska testerna så måste resultaten av de statiska testerna vara godkända och oftast så är en kombination av de här två metoderna att föredra. Om den ena metoden slopas i en utvecklings och testningsprocess så kan man fråga sig om den andra blir lika effektiv än om båda två användes. (Perry, 2000)

### **2.3.3 Manuell kontra automatisk testning**

Den tredje klassificeringen av testmetoder är om det utförs manuellt eller automatiskt. Det som skiljer dem mest åt är att manuell testning utförs av människor och automatisk testning av datorer. Perry (2000) menar även att någon dock måste skapa och utforma testet som datorn ska köra men själva exekveringen görs automatiskt av datorn.

Även om det kan tyckas smidigt att automatisera så mycket som möjligt av testningen, och utvecklingen överlag, så ska det aldrig glömmas att människans analytiska förmåga också behövs. (Perry, 2000)

### **2.3.4 Mobil kontra stationär testning**

Vi har fått lära oss att mjukvarutestning är av stor vikt inom systemutveckling och detta gäller inte minst för mobila enheter. När vi har att göra med mobila enheter så måste även flera andra dimensioner tas hänsyn till. Olika operatörer, batterilängd, prestanda i avseende mindre ram, lagringsutrymme och processor men även mjukvaruheterogenitet som olika operativsystem och liknande är några saker som måste tas med i beräkningen när testning pågår. (Andrade, R. Da Costa, A. Dantas, V. Marinho, F, 2009)

Ett exempel på skillnad i mobila enheters mjukvarufunktioner från traditionella enheter är att om person t ex sysslar med något och får ett inkommande samtal skall det som görs, sparas till samtalet är över så att han inte behöver börja om från början igen. Samma sak gäller om den mobila enheten kraschar eller om batteriet tar slut. Dessa och många andra faktorer som gör det mobila enheternas användande specifikt gör att testprocessen måste anpassa därefter. Alla testmetoder som vi tar upp går att utföra på både stationära och mobila enheter. (Andrade et al, 2009)

Utvecklingen av mjukvaran borde ske omväxlande med testningen, så att en färdigprodukt inte testas på många

områden utan delarna efterhand de blir klara i så stor utsträckning som möjligt. Dock bör det både testas i emulatorer och i mobila enheter eftersom tester i emulatorerna kan missa vissa enhetsspecifika egenskaper. Vissa egenskaper skall testas i enheten på så kallade fälttest, valet av typ av testning i detta sammanhang är av stor betydelse. Användbarhetstest bör göras som fälttest och inte i emulatorer. (Andrade et al, 2009)

Det är även viktigt att testresultaten rapporteras kontinuerligt, detaljerat och framför allt på ett sätt som utvecklarna kan ta del av informationen på ett för syftet gynnsamt sätt. Detaljerat i avseende hur ofta problemet uppstår, i vilka sammanhang det uppstår och liknande för utvecklaren viktig information. (Andrade et al, 2009)

## **2.4 Testmetoder**

### **2.4.1 Stress testing**

När mjukvaran ska påfrestas så mycket som möjligt så kallas det för ”stress testing”. Testaren ser ifall mjukvaran eller systemet klarar av att utsättas för större volymer av saker och ting är vad som anses normalt. Till exempel så testas det så att diskutrymmet klarar påfrestningar i form av programfiler i stora format. Mjukvaran ska strukturellt klara av hanteringen av stora mängder data. Stress testing är något som används när det finns en osäkerhet om hur mycket arbete som mjukvaran klarar av utan att svikta. Syftet med denna typ av testning är att överbelasta mjukvaran eller systemet så mycket som det är möjligt. (Perry, 2000)

### **2.4.2 Recovery testing**

Recovery, eller återställande, är förmågan för en applikation eller mjukvara att kunna starta om fastän det till exempel har kommit till skada. Syftet med denna sorts tester är att även om något oväntat inträffar och mjukvaran kraschar så ska den ändå kunna användas efter en omstart eller återställning. Även om programkrascher alltid bör minimeras programkrascher och förlust av data så är det svårt att eliminera problemen helt och hållet. Det är något som alltid bör ha i åtanke att det kan hända, därför är just sådana här tester väldigt viktiga. (Perry, 2000)

### **2.4.3 Security testing**

En annan metod eller teknik som faller in under strukturell testning är security testing, eller testning av säkerheten i mjukvaran. Det första som görs är att se hur stor säkerhetsrisken är och hur stora de potentiella förlusterna av data är i samband med de hittade säkerhetsriskerna. Som användare kan det tyckas att säkerhet alltid bör vara högt prioriterat men det finns också en grad av värde och betydelse då vissa saker och data prioriteras högre. Hur stor prioritering säkerheten i mjukvaror har i mobila enheter kan variera. (Perry, 2000)

### **2.4.4 Requirements testing**

Under denna typ av testning, kravtestning, så är målet att se så att systemet eller mjukvaran utför sina funktioner korrekt, även under en längre tid. Enligt Perry (2000) så kan den här typen av tester exekveras under



hela testprocessen, men det är svårt att utföra dem innan mjukvaran är funktionsduglig överhuvudtaget. Ofta är det de olika kundernas krav som följs och ses till så att de uppfylls. Det är trots allt kunden som i slutändan ska tillfredsställas. (Perry, 2000)

#### **2.4.5 Regression testing**

När en mjukvarutestare utför regressionstester på mjukvaran så betyder det att han gör om ett test, även om resultatet förra gången var positivt. Det låter kanske besynnerligt, men om utvecklare gör stora ändringar i en mjukvara så behövs det göras regressionstester för att se till så att den nya versionen av mjukvaran har bibehållit stabiliteten som fanns i den förra versionen (Burnstein, 2003). Det räcker inte med att bara testa det som har ändrats i koden. Mjukvara måste testas, i princip, allting om och om igen för att se till så att ingenting har fallerat efter den senaste uppdateringen eller ändringen. Detta är regressionstest. En mjukvara hinner modifieras väldigt många gånger under utveckling så därför är regressions tester av väldigt stor vikt och något som utförs under hela processen. Även om en utvecklare har gjort specifika ändringar i koden och mjukvaran så kan en helt annan del drabbas av fel. Om tiden inte räcker till för att göra regressionstest efter varenda ändring i mjukvaran så måste det noga bestämmas, med hänsyn hur mycket som riskeras i form av förlust av data och dylikt, när de ska utföras(Perry, 2000).

#### **2.4.6 Error handling testing**

Ett mål med en mjukvara är att den ska vara så felfri som möjligt men det är otroligt svårt, om inte omöjligt. Ibland kan dock fel bero på användaren och då behöver mjukvaran kunna hantera detta på ett bra sätt. En användare kan till exempel skriva in fel data i applikationen som helt enkelt inte stöds eller ska stödjas av mjukvaran. Utvecklarna måste alltid räkna med att användaren också kan göra fel. Dessa tester bör som många andra utföras genom hela utvecklingsprocessen. (Perry, 2000)

#### **2.4.7 Heuristic testing eller exploratory testing**

Heuristic testing är en testmetod där testaren blir tilldelat ett område i mjukvaran där han fritt får bestämma hur och vad han ska testa. Det innebär att sitta med mjukvaran och prova olika funktioner utan riktlinjer om hur och vad. Det är upp till testaren att fritt försöka hitta fel genom att på olika sätt ta sig fram i mjukvaran. Metoden är regelfri och har inga riktlinjer. Den bygger på sunt förnuft och kvalificerade gissningar. Metoden bygger på testarens erfarenhet. (Perry, 2000)

## 2.5 Ramverk

För att få en bättre överblick över hur testningen ser ut i de olika utvecklingsmiljöerna har vi ställt upp en tabell med de olika testmetoderna och testkategorierna, där y-axeln visar vilken kategori testmetoderna tillhör och x-axeln visar de olika testmetoderna. De olika kategorierna testning till vänster i tabellen är ställda parvis efter varandra. Vi kan se i tabellen att en testmetod kan till exempel både vara manuell och automatisk, det är alltså inte uteslutande för de olika metoderna att de tillhör en kategori eller en annan. Denna tabell kommer vi fortsättningsvis i vår undersökning att använda för att kartlägga på ett överskådligt vis hur testningen fungerar. Detta gör vi genom att undersöka hur de olika kategorierna och testmetoderna används i agil utveckling kontra vattenfallsutveckling.

	Stress testing	Recovery testing	Security testing	Requirements testing	Regression testing	Error handling testing	Heuristic testing
Strukturell	√	√	√				√
Funktionell				√	√	√	√
Dynamisk	√	√	√	√	√	√	√
Statisk			√	√			
Automatisk	√	√	√	√	√	√	
Manuell	√	√	√	√	√	√	√
Stationär			√	√	√	√	√
Mobil	√	√	√	√	√	√	√

Tabell 1: Tabell över testmetoder och testkategorier

## 3 Metod

### 3.1 Undersökningsstrategi

Vi har valt att göra ett antal intervjuer med beslutsfattare inom organisationen eftersom vår målsättning är att skapa en insikt i aktörernas agerande och i det sammanhang de befann sig i mer än siffror och statistik. Dessutom vill vi få fram en förståelse för problemområdet där det mer ses på mönster, beslut och beteenden istället för få fram ett mätbart resultat i siffror (Trost, 1997). Detta förhållningssätt ger oss en bättre chans att lyfta fram och på ett djupare plan förstå olika parterns åsikter och uppfattningar, och därmed ge oss en bättre möjlighet att uppnå vårt syfte än vad ett mer kvantitativt förhållningssätt till problemområdet skulle kunna göra. Våra frågeställningar utgår även i hög grad från ”vad”, ”varför” och ”hur” frågor vilket innebär att intervjuer av utvalda individer på företaget passar väl för undersökningen vilket är att rekommendera när pågående händelser ska studeras där relevant omgivande faktorer och händelser inte kan kontrolleras (Jacobsen, 2002), vilket var fallet med vår undersökning.

### 3.2 Undersökningshjälpmedel

Vid genomförandet av intervjuerna använde vi oss av en intervjuguide som vi sammanställde för att få svar på våra undersökningsfrågor som är kopplade till undersökningens syfte vilket också rekommenderas av Jacobsen (2002). Vi använde oss av ett antal huvudfrågor eller teman där vi försökte att styra intervjuerna åt ett håll som gagnar vårt syfte. Dessa olika frågor och nyckelord är kopplade till den teori och de kriterier vi hittat för dels mjukvarutestningsmetoder men även utvecklingsmetoder och beslut som tas i samband med detta.

Vi utformade först en intervjuguide (bilaga1) som användes vid de inledande två intervjuerna. Efter en genomgång av resultatet i dessa två intervjuer identifierade vi att den förändring som har skett inom organisationen rör införande av ett mer agilt arbetssätt och med det omarbetade vi intervjuguiden (bilaga2) så att vi kunde få ut mer information kring detta. De två intervjuguiderna användes på samma sätt under samtliga intervjuer och innehåller i stort sätt samma frågor. Vi utformade en tredje intervjuguide till testarna (bilaga3). Vi utformade till sist en komplimenterande fråga som vi mejlade till alla respondenterna och frågade om vi kunde ringa upp dem och ställa (bilaga4). Som följd av detta har vi fått reda på vad de olika respondenterna har för tankar och idéer i de specifika frågor som vi ville ha svar på och det underlättade vårt analysarbete.

#### 3.2.1 Intervjuguiderna

Vi kom fram till tidigt att vi ville ha två stycken olika intervjuguides, en för beslutsfattaren (bilaga 1) och en annan för testaren (bilaga 3). Framförallt p.g.a. de skilda arbetsuppgifterna de har men även för att de har olika insyner i företaget de arbetar på. Vissa frågor är dock lika i båda guiderna för att vi vill se hur likt eller olik de svarar. Anledningen att vi intervjuade beslutsfattare är för att de har satt upp riktlinjerna till hur det skall testas och varför medan en enskild testare kan sitta och utföra vissa tester medan en annan utför andra. Efter vi hade intervjuat två beslutsfattare modifierade vi den intervjuguiden till de resterande tre intervjuerna. Detta för att

efter de två inledande intervjuer så märkte vi att vissa frågor saknades och andra var överflödiga så modifieringarna resulterade i (bilaga 2) för att få ett mer användbart resultat. Slutligen upptäckte vi att vi hade förbisett en fråga som vi ville ställa till alla (bilaga4). Vi kommer nedan att gå igenom fråga för fråga och förklara hur vi kom fram till dem.

I alla intervjuer bad vi respondenten att först presentera sig kort, berätta vilka arbetsuppgifter han/hon har samt hur länge han/hon varit på företaget. En som har varit på företaget i en månad har garanterat mindre erfarenhet av hur det går till än en som arbetat där i fem år och vi tyckte det var intressant att se skillnaderna där.

*”Kan du beskriva hur utvecklingsprocessen ser ut på ert företag?”*. Denna fråga ställde vi till alla beslutsfattare som vi intervjuade då den är självklar för vår undersökning. Vi bad även respondenterna berätta om hur deras utvecklingsmodeller sett ut de senaste 5-10 åren, varför de gjort förändringar samt framförallt hur testningen påverkas av detta. Det var här som vi kände att vi behövde ändra lite i intervjuguiden för beslutsfattarna. Vi märkte att båda de två första intervjuade beslutsfattarna pratade mycket om hur agila utvecklingsmodeller har präglat och påverkat företaget de senaste åren, så vi bestämde oss för att utveckla ett par av våra frågor och be respondenterna att berättade mer om detta.

*”Vilka testningsmetoder använder ni er av just nu, och varför?”*. Här ville vi ta reda på vilka av de vanligaste testmetoderna, enligt litteraturen vi använt oss av, som används i organisationen. För att vi skulle få ut mer av det så frågade vi när i utvecklingsprocessen som respektive metod används samt vilka faktorer som påverkat valet av testmetoderna. Vi fortsatte på samma spår genom att fråga respondenterna om det finns testmetoder som tidigare använts men inte längre, varför det är så samt om de känner till testmetoder som kan tänkas vara relevanta för organisationen. Vilka förändringar som gjorts i företaget gällande testmetoder, vad syftet med dessa förändringar var och om målen uppnåddes frågade vi också eftersom det också är relevant för resultatet i vår undersökning.

För att ta reda på hur mycket som den enskilde testaren själv får bestämma vilka testmetoder som han vill använda så frågade vi även om detta. Denna fråga kom vi att ställa till både beslutsfattare och testare för att se om det blev några avvikelser i svaren.

För att få en inblick i hur det går till när viktiga beslut tas angående testningen så bad vi beslutsfattarna att berätta för oss hur beslutsprocessen såg ut senast de deltog i en sådan. Vi ville bland annat veta hur det väljs vilka metoder som ska användas, vem som tar besluten och hur proceduren går till, från början till slut.

*”Har det blivit någon skillnad i omfattning av mobil kontra stationär testning i företaget när ni jobbade enligt vattenfallsmodellen eller när jobbar agilt?”*. Vi ville få reda på om det hade blivit en skillnad i omfattning av mobila eller stationära tester i vattenfallsutveckling kontra agil utveckling.

Intervjuguiden som vi använde oss av när vi intervjuade testarna var lite annorlunda än de vi använde till beslutsfattarna. Fokus på dessa intervjuer låg framförallt på deras uppfattning om hur självständig en testare kan och får vara. *”Kan du själv välja hur du skall testa, om ja beskriv när och hur?”*, *”Hur mycket i procent skulle du säga att du bestämmer och hur mycket är bestämt?”*. Svaren vi fick av dessa frågor jämförde vi med

den motsvarande frågan som vi ställde till beslutsfattarna för att se om åsikterna var delade eller inte.

För att gå vidare med dessa två frågor så frågade vi testarna om de känner att de skulle kunna bestämma mer, men kanske väljer att inte göra detta. Vi ville veta hur självständiga testarna känner att kan och får lov att vara och hur de känner om detta.

### **3.3 Datainsamling**

Vi har intervjuat fem personer som jobbar med mjukvarutestning på en beslutande nivå inom företaget i fråga samt tre testare. Vi hade för avsikt att undersöka hur testningen fungerar i agil utveckling hos ett företag som tidigare använt vattenfallsutveckling. Som intervjuform har vi använt oss av semistrukturerade intervjuer vilket enligt Jacobsen (2002) innehåller ett visst antal teman med förutbestämda frågor. I vårt fall karakteriserades dessa teman, frågor och sådant som berörde själva mjukvarutestningsmetoderna och den utvecklingsmiljö de används i dels beslutssituationer. De personer som intervjuades har också haft en möjlighet att förklara centrala faktorer som berör ämnet samt att svara relativt fritt under intervjun. På så sätt har vi kunnat följa upp de ämnen som intresserar oss och forma intervjuerna på ett sätt som gagnade vårt syfte. Men vi ville ändå försäkra oss om att ämnet följs vilket vi gjorde genom att till huvudfrågorna skriva ett antal nyckelord eller hjälpord som är kopplade till de olika huvudfrågorna för att på ett enkelt sätt leda in respondenten på det spår som vi är intresserade av. Intervjuerna spelades in på ljudfil så att vi under intervjutillfällena kunde koncentrera oss på ämnet och dynamiken i intervjun vilket även rekommenderas utav Jacobsen (2002). På vår telefonfråga(bilaga4) svarade två testare och två beslutsfattare, men eftersom deras svar var så liknande ansåg vi det svaret tillförlitligt.

### **3.4 Analys av material**

Ljudinspelningar som gjordes under intervjuerna transkriberades och efter att vi hade transkriberat intervjuerna så anonymiserade vi dem för att det inte skulle vara möjligt att se vilket företag som undersökningen är gjord på. Anonymiseringen gjorde vi på intervjuerna med beslutsfattarna då detta var ett krav från företaget där vi gjorde vår undersökning. Processen att anonymisera innebar att ta bort det som inte var relevant för vår undersökning men kunde avslöja var undersökningen är gjord. Vi ställde upp varje intervju i frågor och gick igenom texten för att kunna ta bort överflödiga information som inte behövdes till analysen av materialet. Efter att ha tvättat bort all text om avdelningar på företaget och annat internt snack så skickade vi in transkriberingen i redigerat format för godkännande av företaget. Vi var noga med att ta bort saker som kunde avslöja på vilket företag undersökningen är gjord men för den delen inte ta bort information som spelar roll för vårt resultat. När företaget hade godkänt våra anonymiserade transkriptioner satte vi oss var och en för sig med det insamlade materialet och analyserade teman och områden för att sedan jämföra våra analyser för att sammanställa ett gemensamt resultat. Vi hittade ett antal teman och tog ut nyckelord som de olika respondenterna hade sagt, utifrån de nyckelorden tog vi sedan fram vårt resultat. (Jacobsen, 2002)

### **3.5 Etik**

I vår undersökning följer vi principen för informerat samtycke (Jacobsen, 2002) där informanterna ger sitt samtycke att delta i vår undersökning. Vi informerade respondenterna om vad undersökningen går ut på samt att de som blir intervjuade hade så klart rätt att dra sig ur om de kände för detta. Respondenterna är även medvetna om att deras anonymitet i den slutgiltiga uppsatsen är garanterad.

## 4 Resultat

### 4.1 Den gamla utvecklingsmodellen

I alla intervjuerna som vi gjort så har det framgått att det har skett ett skifte i utvecklingsmodell de senaste åren. Den modell som de har haft innan skiftet var en typ av vattenfallsmodell.

*”Det var en vattenfallsmodell, x antal månader för att specifikera upp 'features' allting som skall in, låt oss säga att det är ett år i förväg.”* Beslutsfattare 2

Det har dock aldrig varit på så sätt att de har haft en utvecklingsmodell som är identisk med en modell från en bok. När respondenterna berättar att de har, eller har använt sig av en vattenfallsmodell så påpekar de att modellen snarare är baserad på vattenfallsmodellen som Görling (2009) beskriver. De har modifierat den så att den passar deras organisation och alla olika avdelningars behov. Det som utmärker vattenfallsmodellen vilket även visade sig i våra intervjuer är att de utvecklade i en bestämd ordning och möjligheten att gå tillbaks ett steg fanns inte utan då fortsatte processen in i nästa steg. Precis som ett vattenfall som rinner i en riktning och inte kan vända och rinna tillbaks. (Görling, 2009)

Så här beskriver en beslutsfattare utvecklingsmodellen företaget hade innan det övergick till agil utveckling:

*”Innan dess hade vi en modell som heter ”heart beat” modellen, som innebär att man har två hjärtslag per år då det kommer ut produkter en till julhandeln och en till sommaren egentligen.”* Beslutsfattare 1

Det visar på två vattenfall där ett har ett slut till sommaren och det andra ett slut vid julhandeln. Målet var att vara färdig med produkter vid två tillfällen per år alltså skall vattenfallet ha runnit ner till dess. Alla vi intervjuade var överens om att det som rådde tidigare var vattenfallsutveckling. (Görling, 2009)

### 4.2 Testning i gamla utvecklingsmodellen

Görling (2009) beskriver att testningen kommer in sent i vattenfallsmodellen. Detta var även fallet på företaget vi gjorde vår undersökning på vilket beskrivs i följande citat:

*”Det betyder att vi måste ändra vårt tänk ganska mycket, förut har vi haft stora block systemtester i slutet som egentligen är kanske tiotusen testfall mot att nu köra minimalt med testning vecka efter vecka och välja lite smartare ”test scope”.”* Beslutsfattare 2

Det testades stora block i av utvecklingsprocessen när mjukvaran skulle vara klar istället för kontinuerligt som i agila utvecklingsmodeller (Avison och Fitzgerald, 2006). Vi såg även att de arbetade som två separata organisationer, testning och utveckling som satt på olika ställen och jobbade var och en för sig.

*”Vi hade dålig kvalitet helt enkelt. Det fanns ingen som hade ett helhetsansvar, utvecklarna ville bara leverera och skita i kvalitet och testarna stod liksom och försökte stoppa.”* Beslutsfattare 1

Det var testare som fick ta emot mjukvara från utvecklare som i och med överlämning till test frånsade sig ansvaret. Vi ser tydligt hur det är två separata instanser utvecklare och testare där båda försöker skjuta över ansvaret på den andra. De jobbar inte som ett lag utan som två separata instanser.

### 4.3 Den nya utvecklingsmodellen

Företaget var på väg in i en mer agil utvecklingsmodell som på vissa avdelningar hade börjat implementeras och på andra var på väg att implementeras.

*”Ja, definitivt. I den här nya utvecklingsmetodiken som tas fram... Hela den metodiken, kontinuerlig utveckling, är ju i princip och i grunden agil.”* Beslutsfattare 3

Det finns olika agila utvecklingsmodeller som skiljer sig på vissa punkter men framförallt har väldigt mycket gemensamt (Cockburn, 2002) och vi har i vår undersökning inte försökt identifiera vilken agil modell som används utan konstaterat att det är en agil utvecklingsmodell som företaget håller på att implementera, vilket alla var överens om.

*”Agila metoder använder vi oss av, framförallt i de här featureteamen då, alltså om vi pratar Agil/SCRUM så, ja...”* Beslutsfattare 4

Eller som en annan beslutsfattare beskriver:

*”Det är utvecklare, testare, managers, projektledare och allt sådant där i ett team. Då sitter dom tillsammans och utvecklar och då jobbar dom ju något ”SCRUM” liknande.”* Beslutsfattare 4

Här ser vi att företaget har gjort förändringen som vi beskriver med att testare och utvecklare sitter ihop i ett lag och arbetar tillsammans. (Petersen K, Wohlin C, 2010)

Som vi har beskrivit i teorin finns det ett antal agila utvecklingsmodeller och på företaget vi undersökte hade de anpassat beroende på behovet på de olika avdelningarna och tagit delar ur de olika modellerna så att det passade deras behov. Extremprogrammering och SCRUM-liknande modeller var det som användes huvudsakligen berättar en av respondenterna. En annan respondent poängterar dock att utvecklingsprocessen ständigt förändras, och utvecklingsmodellen är i ständig förändring.

Alla vi intervjuade var överens om att den nya utvecklingsmodellen som började införas på företaget var agil men inte exakt en agil metod utan en blandning. En beslutsfattare kallade den ”SCRUM:ish” vilket visar på att företaget tar till sig av det agila tänket men när det skall implementeras i en befintlig verksamhet så anpassas den.



Beslutsfattarna försöker också hela tiden få utvecklarna att tänka större och längre än sitt eget område. En utvecklare har sitt område som den jobbar med, men som vi nämnt tidigare så är produkterna betydligt mer komplexa idag och många funktioner interagerar med varandra. Detta sätter högre krav på utvecklarna och utvecklingen (Da Lucia et al, 2008).

*“När man utvecklar en feature är det lätt hänt att man zoomar in på sin feature isolerat, hur funkar den, men du tänker inte hur den interagerar med, om värdfunktioner eller hur de interagerar med din feature. Det är väl någonting vi försöker trycka så att man tänker system redan från början...”*

Beslutsfattare 4

#### 4.4 Testning i den nya utvecklingsmodellen

I och med över gång till en agil utvecklingsmodell så påverkar det testningen också. Istället för att testa i slutet av utvecklingsprocessen så testas det kontinuerligt genom utvecklingsprocessen. Gränserna mellan vad som är utvecklare och vad som är testare suddas ut och flyter ihop. Förr satt testarna för sig och utvecklarna för sig men nu sitter de ihop och jobbar tillsammans.

*”Ja, det här har vi hållit på att arbeta fram i ett år ungefär. Nu finns det bara en utvecklingsorganisation med testare och utvecklare i samma.”* Beslutsfattare 1

Det är ett samarbete mellan utvecklare och testare. Utvecklaren berättar för testaren vad det är för något som han ska implementera i mjukvaran, så att testaren samtidigt kan skriva testfall. Både automatiska och manuella testfall skrivs. När denna ”sprint” är färdig, som beslutsfattare 1 kallar processen för, så godkänns den av projektledaren och nästa sprint påbörjas. Detta visar på SCRUM där det arbetas i sprintar. (Cockburn, 2002)

Det är ett agilare sätt att utveckla och testa på där testare och utvecklare inte åtskiljs (Cockburn, 2002) (Eriksson, 2004). Det innebär även att det testas från början av projektet istället för att vänta till cykeln har nått till testning som i vattenfallsmodellen:

*”Jag skulle nog vilja säga att vi, dels trycker vi testningen tidigare i projektet, dels är det att vi vill att på funktionell utveckling ska man tänka systemmässigt.”* Beslutsfattare 4

Vilket gör att det testas efterhand, veckovis, hela vägen och får inte en stor överraskning när det kommer till att testa om mjukvaran fungerar. Den andra delen av meningen visar på att det utförs med strukturella tester, där det intressanta är hur hela systemet reagerar vid en förändring snarare än hur en funktion reagerar (Burnstein, 2003).

Testarna har även fått en större frihet än tidigare. Heuristic eller exploratory testing där testaren själv bestämmer vad som testas och hur (Perry, 2000). Det är upp till varje testare att testa det han känner behöver ägnas mer tid åt.

*”Men t ex när man kör heuristic testing så kanske man märker att en viss feature inte helt fungerar som den ska och då tar man ju beslutet själv att testa den lite extra kanske. Detta uppmuntras så klart och det är bra.”*  
Testare 1

#### **4.4.1 ISO-9126**

Företaget använder sig av den internationella ISO-standarden för mjukvarutest.  
(<http://www.sqa.net/iso9126.html>, 2009-11-22):

*”ISO-9126, det är kvalitetsattributen. För att testa usability så måste du ha en viss typ av testmetodik, för att testa reliability måste du ha en viss testmetodik, för att testa efficiency så måste du.. Ja ni förstår. Och vi har i dagsläget inte formell spårbarhet mot dom attributen, men vi har täckning som täcker i princip alla. Så att vi har stress tester, vi har duration tester, vi har performance tester, vi har usability tester, vi har rena funktionella tester, vi har performance tester”* Beslutsfattare 3

Respondenterna berättade att denna standard efterföljdes på företaget och det var de attribut som avgjorde hur det testades och vad. De sex områdena som ISO-9126 skall testa är funktionalitet; tillförlitlighet; användbarhet; effektivitet; underhållbarhet och portabilitet vilket de olika testmetoderna sedan skall ta hand om. Organisationen har de övergripande kraven att de sex områdena skall ha testats och de görs med olika testmetoder.

*”Vi har andra sorters ”launchkriterier” som vi bygger upp enligt ISO-kvalitetsattributen. Functionality, reliability, efficiency, usability, portability och maintainability.”* Beslutsfattare 2

#### **4.4.2 Strukturell kontra funktionell testning**

De två olika klasserna funktionell och strukturell testning där funktionell testning är när funktioner specifikt testas medan strukturella är mer övergripande testning av systemet i ett större sammanhang (Burnstein, 2003), tillämpas parallellt med varandra. Syftet med detta var enligt en respondent att få utvecklarna och testarna att tänka ”systemmässigt”. Det är viktigt att se hur de olika funktionerna fungerade i systemet och att de kunde interagera med helheten.

*”När man utvecklar en feature är det lätt hänt att man zoomar in på sin feature isolerat, hur funkar den, men du tänker inte på hur den interagerar med värdfunktioner eller hur de interagerar med din feature”* Beslutsfattare 4

De behövde inte arbeta så här tidigare, då antalet ”features” och applikationer inte var lika många. Produkterna blir komplexare och innehåller fler och fler funktioner och applikationer vilket ställer högre krav på testningen påpekar en respondent.

*”Tidigare behövde vi bara testa enklare saker men produkten har blivit mer komplex och det sätter mer krav på testningen.”* Beslutsfattare 1

Flera respondenter berättade att den stora förändring som övergången från vattenfallsutveckling till agila metoder berodde på behovet att skifta fokus från funktionell till strukturell testning.

#### **4.4.3 Dynamisk kontra statisk testning**

Mjukvaran testas statiskt, det vill säga att koden går igenom för att testaren skall se var felet ligger. Större delen av testen är dock dynamiska alltså test hur mjukvaran uppför sig när koden körs. (Perry, 2000)

*”Kodanalys gör vi också med statistiska verktyg som går in och tittar på koden, finns det något..., är det något som är felkodat och du inte hittar det med dynamisk testning... du kanske inte ser det med dynamisk testning men du ser det med statistiska verktyg.”* Beslutsfattare 4

Statisk testning används som ett komplement till dynamisk testning, huvudsakligen tidigt i utvecklingsprocessen eller när de dynamiska testerna inte räcker till. Detta var fallet såväl i vattenfallsutveckling som i den nya agila utvecklingsmiljön berättar respondenterna.

#### **4.4.4 Manuell kontra automatisk testning**

En av de största förändringarna som genomförts de senaste åren på företaget vi undersökte är att företaget strävade efter att automatisera så mycket av testningen som möjligt.

*”Sen har vi ett stort fokus på interfacetest och automatisering. Det är det som sker nu. Det är mycket krut på det. Vi ska helt enkelt automatisera.”* Beslutsfattare 1

Vi märker en tydlig trend i samtliga intervjuer som vi genomförde med beslutsfattarna. Det är betydligt större fokus på automatisering än tidigare men det betyder inte att de inte testar manuellt påpekar en av respondenterna.

*”Allt kan helt enkelt inte testas automatiskt utan manuella kommer alltid att behövas.”* Beslutsfattare 1

Det är essentiellt att inte ha en övertro på automatiska tester, för den mänskliga hjärnan är fortfarande i vissa avseenden överlägsen menar en av respondenterna. Testfall där t ex stora mängder data ska överföras och när testare vill ”stressa” produkten är gynnsamma att automatisera. Att göra samma sak med tester som involverar UI:t är svårare. Det är lätt att föreställa sig en framtid där alla tester är automatiserade, men intrycket vi får är att manuella tester kommer alltid att finnas och behövas.

Ett annat problem som tas upp i diskussionen runt automatiska tester är att de tar väldigt lång tid att göra. Om någonting ändras i UI:t, vilket sker kontinuerligt under utvecklingen så är det skapade automatiska testet

oanvändbart.

Ekonomi är en aspekt som bestämmer om testning ska automatiseras eller inte. I många fall tjänar organisationen på att automatisera, men ibland är det inte ekonomiskt försvarbart. Det beror helt enkelt på vilka sorts tester som ska utföras. Som en av respondenterna påpekar med följande citat är det ekonomin som styr mycket.

*”När allt kommer omkring så handlar det om att göra vinst.”* Beslutsfattare 5

#### **4.4.5 Mobil kontra stationär**

Våra intervjuer visade att det inte har skett någon förändring i förhållandet mobil kontra stationär testning. Stationär testning är test utförda i en stationär enhets emulator medan mobil testning utförs i den mobila enheten som mjukvaran är avsedd för (Andrade et al, 2009). Alla respondenter var ense om att i början av processen när mjukvaran bara existerar stationärt och inte fungerade mobilt så testades mjukvaran självfallet stationärt. När mjukvaran kan exekveras på mobila enheter så testas den med fördel mobilt.

*”Det har att göra med när i processen man befinner sig. I början testas det stationärt men mot slutet testas det nästan enbart mobilt. I början av processen kan man inte testa mobilt då mjukvaran inte finns i mobilen ännu.”* Beslutsfattare 1

Alla respondenter var eniga om att förhållandet mobil kontra stationär testning inte skiljer sig i agil utveckling från vattenfallsutveckling på företaget i fråga.

#### **4.4.6 Testmetoderna**

Vi frågade beslutsfattarna hur många av de sju testmetoderna som vi beskrivit i teorin som de använde. Alla används, mer eller mindre, men error handling test var en testmetod som ibland fick stryka på foten.

*”Alla testmetoder som ni har på er lista använder vi oss av, förutom error handling testing som vi använder, men inte särskilt mycket. Det finns inte tillräckliga syften för att utföra det mer än vad vi gör.”* Beslutsfattare 2

Som Perry (2000) beskriver error handling, eller negativ testning som det också kallas så är det att testare avsiktligt försöker injicera fel i mjukvaran för att se hur den hanterar det. Flera av våra respondenter såg denna typ av test som ett tidskrävande och inte så effektivt sätt att testa på.

En annan informant tyckte att de inte använde recovery test (Perry, 2000) som innebär att starta om efter en krasch med minimal förlust av data.

*”Nej, jag tror inte, eller jo, kanske typ som recovery till exempel, där tror jag spontant att vi är lite svaga.”*  
Beslutsfattare 4

De respondenter som vi har intervjuat arbetar på olika avdelning och detta har en påverkan på vilka testmetoder som används och i vilken utsträckning de används. En respondent berättar om error handling testning och på avdelningen han arbetade på så används den betydligt flitigare än på de andra. Vidare berättade denna respondent att även att denna metod bör användas så tidigt som möjligt i utvecklingen:

*”Error handling handlar om att utmana systemet medan du har tid att fixa det, om du lägger den utmaningen sent då har du inte tid att parera om du skulle hitta något fel.”* Beslutsfattare 4

Requirements testing eller kravtestning (Perry, 2000) som är att testa specifika funktioner eller någonting speciellt som en beställare har efterfrågat är en testmetod som det läggs väldigt stor vikt vid. Beslutsfattare 1 berättar att krav finns från företaget på produkten och att alla funktioner och applikationer i produkten ska fungera. Kunderna beställer egna funktioner eller applikationer i produkten och dessa prioriteras. Anledningen enligt respondenterna till att kundernas test kommer i första hand är att om kundernas applikationer inte fungerar så kommer produkten i retur till företaget för omarbetning. Vissa kunderna gör sina egna testfall berättar en respondent vilket kan försvåra för företaget då de inte är insatta i hur testen är designade.

Alla respondenterna testare som beslutsfattare berättade om Heuristic eller Exploratory testing (Perry, 2000) som är ett fritt sätt att testa utifrån erfarenhet testaren har sedan tidigare.

Självklart är det så. Man blir ju hela tiden bättre på det man gör och man förstår hur saker och ting fungerar. Ju mer rutinerad man är ju snabbare går det också och då kan man kanske ”sväva” lite utanför ramen och köra lite testning på andra saker och ting som man kanske märker behöver lite mer fokus.” Testare 3

Denna testmetod användes på sätt och vis i vattenfallsutvecklingen berättar Beslutsfattare 3. Det fanns inget direkt namn för den och testarna visste inte om att det var heuristic testing de gjorde. Så formellt började företaget använda heuristic testing för ungefär två år sedan. Ungefär samtidigt som utvecklingen på företaget gick över till agila utvecklingsmodeller.

En av beslutsfattarna skiljde sig i sin uppfattning gentemot de andra angående heuristic testing.

*”Heuristic testing betyder inte att testaren kan besluta mer själv utan det beror på var i utvecklingsprocessen man befinner sig. Tidigt i utvecklingsprocessen är det mer bestämt hur man skall testa men ju närmare man befinner sig release desto friare blir testaren i sina beslut.”* Beslutsfattare 3

Alla andra respondenter var av uppfattning att heuristic testing var ett led i att ge testarna mer frihet och få arbeta efter erfarenheter de hade skaffat sig på ett sätt som skiljde sig från de regelbundna testmetoderna.

## 5 Diskussion

Hur fungerar testning i en agil miljö sett i förhållande till vattenfallsutveckling? Många gör denna förändring eftersom komplexiteten i projekten kräver en större dynamik (Da Lucia et al, 2008). Testning i den agila utvecklingsmiljön fungerar som en dynamisk fortgående process som anpassas efter rådande behov. Det finns tydligt föreskrivet i agil utveckling var i processen testningen sker men inget om vilka testmetoder eller typ av testning som skall utföras. Vi ville undersöka den påverkan och se om det var andra faktorer inom testning som påverkades i agil utveckling.

Som vi tidigare har beskrivit så skiljer sig agil mjukvaruutveckling sig markant från vattenfallsutveckling. Ett mer strikt och inrutat och ett mer anpassningsbart och inte så regeltungt sätt att arbeta på (Görling, 2009). Testning är en precis vetenskap i den aspekten att antingen fungerar det eller så fungerar det inte. Det kan så klart fungera mer eller mindre tillfredsställande men ett fel i mjukvaran är alltid ett fel. Sett ur den aspekten så kanske det skulle fungera bäst att följa strikta ramar och förordningar?

Hur ser testning ut i den agila utvecklingsmiljön och hur påverkas de olika testmetoderna och kategorierna av testning? Vi har tittat på hur dessa aspekter ser ut i agil utveckling och satt det mot det tidigare arbetssättet vattenfallsutveckling.

Vi ser på kategorin funktionell kontra strukturell testning där det handlar om att antingen titta på enskilda funktioner eller hur det interagera i ett system med andra funktioner (Perry 2000) var något som våra respondenter berättade en hel del om. Att testa hur hela systemet fungerar blir essentiellt för kvaliteten och får mycket större plats i komplexare system. Självklart måste de enskilda funktionerna fortfarande testas men det spelar ingen roll om de fungerar var och en för sig om de inte kan interagera i ett system.

Det har med mjukvarans utveckling och komplexitet att göra som vi tidigare har tagit upp (Da Lucia et al, 2008). Den tilltagande komplexiteten av mjukvaran kan tänkas ställer högre krav på alla aspekter av utveckling och testning, det säger sig nästan självt. Att företag och utvecklarna där i måste anpassa sig till detta är av stor vikt och kan säkert vara en stor utmaning. Vi användare önskar oss fler och mer avancerade program och applikationer till våra mjukvaruenheter som smartphones, surfplattor, datorer och vad vi kan tänkas använda.

En annan kategori är dynamisk kontra statisk testning där det handlar om att antingen göra kodanalyser som i statisk testning eller exekvera koden (Perry, 2000). Båda används men det föreföll att den dynamiska var mycket vanligare. Det kan tänkas att det har med att den dynamiska testningen ger mer direkta utslag, att man ser vad som fungerar och inte fungerar och den statiska testningen kommer in som ett komplement när testare eller utvecklare söker efter specifika fel i koden. Att sitta och titta i kod för att försöka hitta fel utan att veta att det finns ett fel är ett sisyfosarbete utan dess like, som att leta efter en nå i en höstack.

Nästa kategori är automatiserade testning kontra manuell testning (Perry, 2000). Att företaget ville automatisera känner vi kanske är en tidsenlig trend vad gäller många arbetsområden i vårt industriella samhälle. Det som kan göras automatiskt av någon typ av dator eller maskin istället för manuellt av människor

tar bort monotonin inom många arbetsområden och kan i många fall göras effektivare av en maskin. Automatisering av olika arbetsområden är något som händer överallt och att testning skulle vara undantaget hade varit märkligt.

Vi ser även på kategorin stationär kontra mobil testning där den stationära testning sker i datorer medan den mobila som namnet antyder sker i mobila enheter (Andrade et al, 2009). Det kanske är naturligt att innan mjukvaran kan testas mobilt så testas den stationärt och när den väl är redo för mobila tester så är det att föredra.

De olika testmetoderna har vi också undersökt och sett att vissa faller i skymundan framför allt recovery test och error handling test (Perry, 2000) medan andra prioriteras som requirement test (Perry, 2000). Att ekonomiska aspekter styr mycket inom företag är en självklarhet och att det skulle styra hur företagen testas sin mjukvara kan inte uteslutas. Det kan tänkas att vissa tester är helt nödvändiga för att kunden eller beställaren skall godkänna mjukvaran medan andra områden blir en ”sista saken vi gör om vi får tid”. Vi som konsumenter vet och har upplevt allt för många gånger att vi sitter med mjukvara som inte fungerar hundra och irriterats över detta. Det är något som utvecklare och testare är väldigt väl medvetna om men tid är alltid en faktor och lanseringsdatum kan få företag att tumma på den kvalitativa aspekten.

En annan aspekt av testning är dess förhållande till utvecklingen rent organisatoriskt och fysiskt. Att utvecklarna och testarna satt var och en för sig i var sin organisation och på olika fysiska ställen som de hade gjort på vårt undersökta företag innan övergången till agil utveckling måste försvåra samarbetet. Att sitta ihop i samma organisation och ta ett gemensamt ansvar istället för att skicka mjukvaran mellan varandra med noter om vad som skall göras gör att utvecklare och testare tar ett gemensamt ansvar och inte bara skjuter över problemen. Självklart så måste möjligheten till att kommunicera med varandra direkt påverka processen positivt men även engagemanget i att sitta tillsammans och ta ett gemensamt ansvar. När man sitter som i den agila miljön tillsammans och kan rent fysiskt se varandra och samtala kan tänkas vara effektivare. Det är alltid lättare att skjuta över ansvaret på någon annan om möjligheten finns, ”det har utvecklarna missat” eller ”testarna har inte gjort detta”, men om det är en enhet som har ett gemensamt ansvar för att det som levereras skall fungera på ett tillfredsställande sätt så är det svårare att peka finger. Det är antagligen en mänsklig egenskap med vi och dem mentalitet och när det inte längre finns ett vi och dem så blir det ett lag som jobbar tillsammans istället för mot varandra.

## 6 Slutsats

Vi har undersökt hur mjukvarutest fungerar i agil systemutveckling på ett stort företag. De aspekter vi har granskat är dels vilka testmetoder och testkategorier som används, och dels hur den organisatoriska strukturen av test och utvecklingsorganisationen påverkar testningen?

Den stora förändringen i agil utveckling kontra den tidigare vattenfallsutvecklingen var att utvecklare och testare arbetade ihop och testade mjukvaran efterhand istället för när den skulle vara färdig, eftersom det är en del av det agila sättet att arbeta. Det medförde att testare och utvecklare tog ett gemensamt ansvar för slutprodukten.

Förhållandet funktionell kontra strukturell testning hade förändrats till att företaget testade mer strukturellt. Detta är en naturlig utveckling av att produkterna blir komplexare och får fler funktioner.

Gällande statiska eller dynamiska tester så hade förhållandet inte ändrats i samband med övergången till agil utveckling. Det utfördes båda och i ungefär lika stor utsträckning vilket innebar mest dynamiska tester. Vissa saker testas gynnsammare statiskt men det mesta testas bättre dynamiskt. Statiska tester kommer som ett komplement till de dynamiska. Automatisering av testning var något som företaget vi undersökte eftersträvade numera. Vi kan inte se om det är ett resultat av att företaget har börjat jobba agilt eller om det är något som hade förespråkats om företaget hade stannat kvar i vattenfallsutveckling.

Förhållandet mobilt kontra stationärt testande var inte något som hade förändrats i samband med övergången till agil utveckling.

Den enda skillnaden i testmetoderna som tillkommit i den nya utvecklingsmodellen är att företaget använder sig av heuristic eller exploratory testing. De andra metoderna används i samma utsträckning förr och nu. De hade använt sig av heuristic eller exploratory testing innan den agila utvecklingen men den hade då inte haft en benämning och efter övergången till agil utveckling så förespråkades denna testmetod som är i linje med hur agil utveckling fungerar, friare och inte så styrt.

Recovery och error handling test var de testmetod som kom i skymundan. Det var likadant när de utvecklade enligt vattenfallsmodellen som i agil utveckling. Däremot var requirements test är en högt prioriterad testmetod, både i agil utveckling och i vattenfallsutveckling. Att företaget hade testat de krav som kunden har på produkten är av vikt eftersom det är vad kunden kommer att titta efter, det är ju krav kunden har ställt.



## **6.1 Förslag till fortsatt forskning**

Vi fick i vår undersökning inte tillgång till dokument som visade på hur effektiv testning var, vilket hade varit intressant. En undersökning om hur omfattning av testning påverkar kvaliteten till exempel hur utebliven testning behöver kompenseras ekonomiskt genom utgifter för utveckling. En annan intressant aspekt att titta på hade varit hur pass effektivt testningen i agil mjukvaruutveckling är i jämförelse med testning i vattenfallsutveckling. Ytterligare en annan intressant aspekt att forska i hade varit i vilken typ av projekt vattenfallsutveckling är mest lämpad och när agil mjukvaruutveckling lämpar sig bäst.

# Bilagor

## Bilaga A - Intervjuguide 1 med beslutsfattarna

### Presentation

- Hur länge har du varit på företaget?
- Vilken är din nuvarande roll?
- Hur länge har du jobbat med testning?
- Hur länge har du haft en beslutsfattande position?

### Val av testmetoder

- Kan du beskriva hur mjukvaruutvecklingsprocessen ser ut på ert företag? (det kan också vara ett sätt att kolla den utvecklingsmodell intervjupersonerna använder mot den som företaget mer eller mindre har föreskrivit)
- Vilka testningsmetoder använder ni er av just nu och varför? (Vi har en lista av metoder från vår teori plus de metoder som respondenten tar upp i fråga ett ställer vi följdfråga tre och fyra om.)
- När i mjukvaruutvecklingsprocessen använder ni den metoden? Vilka metoder används i respektive fas i mjukvaruutvecklingsprocessen?
- Varför väljer ni den testmetoden? Vilka faktorer är väsentliga för valet?
- I vilken utsträckning kan den enskilde testaren själv bestämma valet av testmetoder?
- Kan du beskriva hur det gick till senast du var med och beslöt om vilken testmetod som används? (Här kommer vi att guida respondenten genom processen genom att ställa följdfrågor om olika beslutssituationer.)
- Vilka förändringar har gjorts vad det gäller testmetoder under de senaste 1-2 åren (eller under den tiden du har arbetat med testning, om detta är en kortare period)? Vad var syftet med dessa förändringar och vad uppnåddes med förändringarna?

### Kompletterande fråga

Har det blivit någon skillnad i omfattning av mobil kontra stationär testning i företaget när ni jobbade enligt vattenfallsmodellen eller när jobbar agilt?

## **Bilaga B - Intervjuguide 2 med beslutsfattarna**

### **Presentation**

- Hur länge har du varit på företaget?
- Vilken är din nuvarande roll?
- Hur länge har du jobbat med testning?
- Hur länge har du haft en beslutsfattande position?

### **Val av testmetoder**

- Kan du beskriva hur mjukvaruutvecklingsprocessen ser ut på ert företag?
- Använder ni er av agila utvecklingsmetoder? Om ja, kan du beskriva hur ni gör detta och på vilket sätt det påverkar testningen?
- Vilka testningsmetoder använder ni er av just nu och varför? (Vi har en lista av metoder från vår teori plus de metoder som respondenten tar upp ställer vi följdfråga tre och fyra om.)
- När i mjukvaruutvecklingsprocessen använder ni den metoden?
- Varför väljer ni den testmetoden? Vilka faktorer är väsentliga för valet?
- Finns det testningsmetoder som ni tidigare har använt men inte använder längre? Om ja, varför?
- Känner du till testmetoder som är relevanta för er men ni inte har använt eller använder er av? Om ja, Vet du varför?
- Vilka förändringar har gjorts vad det gäller testmetoder under de senaste åren (eller under den tiden du har arbetat med testning, om detta är en kortare period) och när gjordes de?
- Vad var syftet med dessa förändringar och vad uppnåddes med förändringarna?
- I vilken utsträckning kan den enskilde testaren själv bestämma valet av testmetoder?
- Kan du beskriva hur det gick till senast du var med och beslöt om vilken testmetod som används?

### **Kompletterande fråga**

Har det blivit någon skillnad i omfattning av mobil kontra stationär testning i företaget när ni jobbade enligt vattenfallsmodellen eller när jobbar agilt?

## **Bilaga C - Intervjuguide testare**

### **Presentation**

- Hur länge har du varit på företaget?
- Hur länge har du jobbat med testning?

### **Val av testmetoder**

- Kan du själv välja hur du skall testa, om ja beskriv när och hur?
- Vad baserar du dina beslut på när du bestämmer vad som skall testas?
- Hur mycket i procent skulle du säga att du bestämmer och hur mycket är bestämt?
- Skulle du kunna bestämma mer själv men väljer du att inte göra det?
- Har det att göra med hur rutinerad man är på hur mycket man som testare beslutar själv?

### **Kompletterande fråga**

Har det blivit någon skillnad i omfattning av mobil kontra stationär testning i företaget när ni jobbade enligt vattenfallsmodellen eller när jobbar agilt?

## Bilaga D - Anonymiserade transkriptioner

### Frågor

Hur ser utvecklingsprocessen ut på ert företag?

### Svar från intervjuobjekt 1

Man säga att processen ser väl ut som så att... Vi har en vad vi kallar "main-branchen" liksom där vi har mjukvaran. Och sen så utvecklar vi ju "features" som sen integreras. Och det kan även då vara någon slags "buggfix" som också integreras. Och sen så efter vi vid ett givet läge så "branchar" man ut... Och sen så här har man någon slags "release-branch" som sen här då liksom... Här är någon slags... Från här och sen ut till kunden liksom.

Och redan här då kanske har man en sådan här "Customer Acceptance" aktivitet. Att man skickar ut till kunden och frågar vad dom tycker om det här och så bollas det tillbaks. Men här uppe är dom som heter ett. Dom har hand mycket om "produkt test". Fokus är på "sales-itemet" och liksom hela produkten med olika "contents" och olika kustomiseringar för operatören medan vi, alltså här nere (pekar på Main-Branchen) är det Software mer och vi håller ju mer på med själva "grundmjukvaran". T ex "Browsersn fungerar". Den kanske inte har ett visst... En viss förinställd att webbläsare ska komma upp först liksom och sådant där men här byggs ju liksom grundfunktionaliteten. Sen finns det lite andra organisationer då som då t ex "Install of Applications" som skapar vissa grejor och så finns det "Content and customization" som gör kustomisering och allt detta går ju liksom in sen till sin produkt. Så vi levererar just också upp till dom och så sätter dom ihop hela produkten.

Så detta är då vår organisation. Det är här jag jobbar (Pekar på "main-branchen") det är den här jag sätter testprocessen för. Man kan väl säga att i stora drag så finns det ett "systemtest team" som sitter på "main-branchen" här och exekverar kontinuerligt testaktiviteter för att skapa en kvalitetsrapport som dom då sen kan förse projektledare och andra "stake-holders" så dom kan ta beslut liksom. Så här är det då inte fokus på att hitta så mycket fel utan det är mer t ex "Se till att vi har en bra kvalitet på "main" så att vi kan göra saker med de, så vi kan ta beslut på det". Det är då någon slags systemtestare som sitter där då kontinuerligt på "main". Och sen så har vi då "features" och "maintenance" som vi levererar och då är det ett så kallat "feature-team" som sitter och jobbar i en "feature". Det är utvecklare, testare, managers, projektledare och allt sådant där i ett team. Då sitter dom tillsammans och utvecklar och då jobbar dom ju något "SCRUM"-liknande.

Och det är då någon slags "feature test" som man börjar med där i "sprintar". Sen efter det har vi vad vi kallar något slags "feature regression test" där vi testar typ "Ok nu är det nästan complete och det verkar fungera. Hur slår det på plattformen?". Då måste man då välja ett "scope" för att... "Vilka moduler har vi varit inne och pillat i? Hur ser våra dependencies ut? Vad kan vi slå på?". Och då kanske man väljer att köra lite testfall som... Om jag nu har implementerat en ny mediaspelare, så kanske jag vill testa lite på "DRM" (Digital Rights Management) och hur det interagerar. Om jag t ex har java i bakgrunden, vad

händer då och så vidare. Och sen så i slutet så har man någon slags "integration test". Man "re-base:ar" kontinuerligt då från "main-branchen" så man hela tiden har det nyaste. För här integreras ju massa nya saker hela tiden så man måste hela tiden "re-base:a" så man ser till så man har det nyaste. Och då precis i slutet som vi då kallar "integration test" som man då kör, där man då "re-base:ar" det senaste och då kör man kanske kort en-timmes test där man då testat det absolut viktigaste för att se att dom här... Att det inte kom in någonting här som sen förstör allting.

Så måste man ha det. Så det är det vi gör här uppe ungefär, men mer än så har vi liksom inte sagt. Det är det ramverk som måste uppfyllas. Sen här när dom levererar koden så måste dom också leverera automatiska testfall till "main-branchen" så man kan "regressions testa" sen här nere kontinuerligt. Sen hur det här efterföljs det är inte alltid klockrent kanske. Men detta är det vi siktar på hela tiden.

Och här nere på "main-branchen" kör man ju då liksom... Alltså alla typer av testning. Vi kör stabilitetstester, prestandetester, funktionella tester. Vi kan säga så här: Vi har ju "feature team" är (pekar på tavlan) och så finns det någonting som vi kallar för "functional area teams" (FAT) som sitter och gör "maintenance". Dom gör alltså "bugg-fixarna".

Dom ger också resurser till "feature-team", så de är så att säga resurs-pooler. Och dom har även markerat vilka testfall i deras område som är extra viktiga. Och de testfallen brukar också systemtestarna köra så dom kör det viktigaste i varje område också. Och sen kör dom "interaction tester" och liknande på hela systemet. "Performance", "Interaction" och den typen av tester.

Men överlag kan man säga att... Det rör sig om testfall som i stor del... Speciellt där jag jobbar då på två där är möjligheterna för att automatisera inte "superbra". Alltså det finns ju "unit tester" och där man testat automatiskt på väldigt låg nivå och det ska ju alla utvecklare använda då för i "feature teamen" så ska dom använda TDD (Test Driven Development) som utvecklingsmetodik liksom. Testdriven utveckling liksom. Där dom ska skriva en massa "unit tester" och sen så utveckla på det och se så att dom passerar.

Men utöver det så ska testarna skriva, kanske mer... För Testdriven utveckling är väldigt fokuserat på... Det är ju en utvecklingsmetodik så det är ju inte så att man skriver alla möjliga test som finns utan då är testarna med där och skriver komplementär testfall för som utvecklarna kanske inte tänkt på.

Men utöver det så finns det inte jättemycket automatiska tester. Att automatisera via UI:t (User Interface) är ju svårt. Vi har något som kallas för BRAT där man kan "script:a" hur den ska göra men problemet är att om UI:t ändras så "pajar" allting. Och det tar lång tid att göra. Det har varit mycket problem helt enkelt. Det börjar dock bli bättre. Det skickas en massa requests "Vi behöver ändra det här, det här och det här" och så blir det sakta men säkert bättre och bättre. Men vi har ju liksom inte hundratals enheter som står och matar automatiska testfall genom UI:t liksom. Det har vi inte. Utan då handlar det mycket om manuella testfall som man kör.

Testfallsbaserat så att säga. Så inte så mycket "Exploratory". Inte så mycket fritt så utan... Framförallt då också eftersom vi har resurser i ett annat land så brukar det vara så att man designar testfallen här och så skickar man testfallen till ett

annat land och så kör dom testfallen där. Nu börjar det ändras dock för nu har dom också fått egna ansvarsområden, men i början var det så. Designen av testfallen är det svåra och exekveringen av dom är det lätta. Kompetensen ligger i designen av testfallen inte av att exekvera dom. Oftast. Sen finns det så klart undantag där testfall är så komplexa att man t ex måste kolla i loggar och sådant där. Men generellt sätt är det så här. Och vi kör ju då inte så mycket "exploratory" där det då ligger kanske mer kompetensen i själva exekveringen.

Så det är lite hur vår utvecklingsprocess ser ut. Sen är det komplext där med hur kraven kommer in och sådant. Det finns en hel process för hur kraven tas fram och hur dom kommer in till "feature team:en" vid olika "feature briefs"... Massa möten och sådant. Men kort sagt så har vi två som leverar plattformen upp till tre.

Alltså dom tar t ex emot plattformen och sen så ser dom till så att den funkar med vår mjukvara. Sen levererar dom då som sagt till Software och säger "Nu funkar plattformen", men så funkar den oftast inte helt. Det är fortfarande en massa problem eftersom dom har testat den på sin nivå sen lägger man ju på lite mer grejor så då börjar det strula. Sen levererar vi det ju till då produktutvecklingen, och då finns ju även "Install of application" som också levererar dit och "Content and customization" som också levererar till två. Och sen leverar dom då till kunden. Så ser det liksom ut.

Så då har vi ju olika ansvarsområden. Dom testar ju på en nivå, vi testar när vi har lagt på våra applikationer, dom där testar sina individuella appar sen leverar dom dom och sen tar dom inte ansvar för dom längre. Utan dom leverar bara sina appar. Dom funkar liksom men om saker och ting förändras så måste vi hitta dom felen i dom applikationerna för dom har liksom redan levererat dom. "Content and customization" levererar sina kustomiseringar och sådant och så kör "PD" en sådan där helhetstestning på allt. "Sales item testing" kallar dom det, där dom testar hela paketet liksom. Sen kan man då fråga sig om det inte blir någon dubbel-testning här emellan då och det är ju komplexiteten i en stor organisation.

Använder ni dessa testmetoder (Lista)?  
Varför och när i processen används respektive metod?

Man kan säga så här: Om vi ska titta lite på vad man gör på "main-branchen"... Där är rätt fast vad vi gör. Ofta på "feature teamen" så lämnar dom det mycket upp till individuella "feature team". Alltså så länge dom kan garantera sin kvalitet så lämnar vi rätt så öppet vad dom behöver göra. Alltså vissa kanske behöver köra "stress tester" och andra behöver kanske inte göra det. Men generellt sätt kan man säga att... Vi har delat upp det så här... Tidigare så hade vi en uppdelning som nu inte gäller riktigt längre. Men innan så hade vi "interface test": det var allting som skedde på interface och under, "funktions test": det var liksom någon slags.. Ja, man testar funktionaliteten via UI:t (User Interface). Och sen hade vi "System test" som var då kanske mer performance och sådant. Så vi delade upp systemtest, det finns ju många olika. Management valde några som de ville fokusera på och det var då: performance, stability som då är reliability men dom kallar det för stability, interactions, stress.. Vad var det mer... Nej det var nog dom fyra som var huvudfokus på. Tror jag. Hur som helst så var det den gamla definitionen vi hade. Det är väl detta som feature-teamen ska använda sig av när dom tänker liksom så här "Ok vad behöver vi testa? Vi har en massa automatiska interface-tester, vi har testat av alla funktionella krav vi har. Har vi testat performance? Ja det har vi. Har vi testat stability? Ja det har vi.

*Interactions? Stress?* Och sen så klart UX (User Experience) och usability. Alltså den typen av testningen har vi då också. Det är lite dom grundläggande... Övergripande ramverk som vi hade innan men det är inget som hindrar "feature teamen" nu från att... Alltså har dom kompetensen inom någonting annat, det kan vara "recovery testing" t ex, vad händer när telefonen kraschar... Om batteriet trillar ut liksom, vad händer med telefonen då? Klarar den av sådana saker? Det kanske behövs (sådana här tester). Andra är t ex "power consumption" som vissa behöver testa men inte andra. Och sen finns det ju... Det är inte så att du måste göra det här, det här och det här utan det är ganska fritt.

Men i t ex systemtest i ett annat land så har vi så kallade "user scenarios" som dom kör varje vecka och det är 22 stycken användar-scenarion, så det är liksom den absolut högsta nivån som dom kör och dom här kör man för att... Dom här 22 scenariosen måste funka hela tiden för att pajar dom så då är det någonting så allvarligt att kunden kommer att märka det. Och utöver det så har vi krav från kunder och beställare på stability som måste uppfyllas om man ska kunna sälja produkter till dom.

Då sätter man igång en massa enheter som kör och dom ska då klara tillsammans t ex 800 timmar eller 1000 timmar eller något sådant där. Så kör man det på ett antal telefoner så att det blir rimligt liksom. Kanske om man har 10 enheter eller något sådant. Så kör man dom 80 timmar var så blir det 800 timmar.

Så låter man dom stå och rulla liksom. Sen har vi ju performance... Dom har typ 300 testfall eller något sådant som man kör varje vecka för att skapa trender för "main-branchen". Så från alla områden har vi samlat ihop 300 testfall som vi kör och så får man trender för det liksom och så kan man följa det över tiden.

Sen har dom ungefär 6000 testfall som dom kör varje två veckor. Om och om igen. Varje två veckor. Det är dom 6000 viktigaste testfallen från alla grupperna som dom har gett till systemtest. Plus lite andra interaktioner och sådant som är systemtäckande. Som dom då kör kontinuerligt. Och det är samma sak här: Dom ska ju funka i princip. Är det något fel här så ska dom säga "*Våra grundtest pajade här. Då måste vi göra en djupare undersökning och så måste vi testa mer här från vår sida*". Det blir som ett litet alarm. En vecka kanske det kommer det in 4 nya features och då kör man dom här testfallen och så pajar 5 testfall här och då går det tillbaks så måste dom göra fixar till features:en.

Och det ska helst vara automatiskt. På den andra avdelningen, där börjar dom ju mer och mer automatisera. Så då blir det ju helt annorlunda. Men dom här är manuella genom UI:t.

Det kan vara mycket sådant här: Minnet fullt, databaser är fulla... Många inputs samtidigt. Man vet att mjukvaran är i ett känsligt state och så försöker man krascha den. Det kör vi rätt mycket. Recovery testing som jag ser är nästa, det kör vi inte alls...

Stress-test ska ju ske under feature-utvecklingen. Så att när dom utvecklas så ska dom köra de här testerna. I "sprintarna" ska dom inkludera den. Så att "*Nu har jag utvecklat ett use-case och då ska jag även ha testfall som stressar det use-case:t*". Om ett use-case är t ex att jag ska lyssna på en MP3-låt så tar utvecklaren fram det men då ska jag som testare fundera ut hur man kan stressa



det här. T ex om jag har minnet fullt när jag kör min MP3-spelare, funkar det ändå? ”Ja det gör det”. ”Ok, bra”. Så det är från början de här testerna.

Vi får inte leverera utan att allting fungerar, så att här ska allting fungera innan du lämnar till ”main-branchen”. När du levererar din ”feature” så ska den fungera helt och hållet. Sen kan det ju vara så att stressen testen är på hela systemet och då kanske man gör dom lite senare när man har implementerat ett sub-system liksom. Du kanske måste implementera ett antal use-cases innan du kan köra vissa stress testfall som går över hela systemet liksom. Och då gör du det lite senare så klart.

Stabiliteten är jätteviktig på telefonen. Och jag menar, man måste ju alltid göra en ”return of investments” på sina testfall. Alltså man funderar på hur stor chans är det att det här händer överhuvudtaget? Jag behöver kanske inte göra mitt absoluta, konstigaste testfall som kan hända 1 gång på 3 miljarder. Det kanske inte är lönt, men att jag har minnet fullt när jag kör min MP3-spelare det kommer att hända alla någon gång. Och ifall det inte funkar så blir return-raten 100% till slut ju. Förr eller senare kommer alla göra det, telefonen kommer att paja, dom kommer bli arga och dom kommer vilja lämna tillbaka den (kunden). Egentligen är ju allting ”return-rate drivet”, i grunden... Sen så är det svårt att se det i realiteten. Men om ingen kommer att lämna tillbaka telefonen p.g.a det här felet och ingen kommer att tycka det är dåligt och det kanske händer 1 gång per 3 miljarder då är det inte lönt att spendera jättemycket resurser på det för det finns mycket annat som vi måste testa och som är mycket viktigare. Så man får tänka så här: *”Ok jag kommer att kunna exekvera 100 testfall och av dessa hundra kommer 5 att vara stress test. Vilka fem ska jag köra?”*.

Det har inte formaliserats så himla mycket och vi har aldrig pratat riktigt mycket om recovery testing. Eller jag pratade om det när jag, för fyra år sedan, när jag skrev det systemtest-dokumentet som vi hade. Och i första utkastet så hade jag med alla olika metoder men då sa managern att det är alldeles för mycket och vi kommer inte ha tid att kunna ta till oss detta så vi måste fokusera på vissa saker istället. Och recovery testing var en av dom som föll bort. Men visst, det kan säkert hända att vissa testare kör något recovery test. Men det är inte så formaliserat som t ex performance testing, stability testing, stress testing och interaction testing. Dom fyra är mer formaliserade. Mer spridda genom hela organisationen.

Ofta sitter arkitekter med security-test istället, det är inte testare som... Alltså t ex vi har en sådan här Digital Rights Management lösning...

Till viss del så testar man kanske och samma sak med antivirus, vi testar till viss del den typen. Där har vi ju security testing. Men jag menar... Dels har vi ett inbyggt system, vi har ju inte öppnat upp större delen av vår plattform tidigare. Så därför har det inte varit samma behov, som på t ex Windows, av security testing. Men till viss mån har det ju varit, som med DRM t ex, då har det varit lite testning som har gjorts men oftast så är det ju arkitekten som har designat det som sen ser var det finns brister. Det finns inte heller så mycket verktyg för att testa det här. Sen är ju virus inte så utbredd. Procentuellt är det väl mindre än 1% som är security testing.

Det är i så fall väldigt tidigt i processen.

Ja det måste det vara.. För du kan ju inte komma sent u processen och ”Oj, här

*har vi ett problem*” för då måste vi designa om hela lösningen från början. Och då missar man *”releasen”*.

Alltså då uppe vid *”release-branchen”*. Vid *”main-branchen”* så fokuserar vi på att få hela funktionaliteten att fungera och sen så finns det då vissa krav som är specifika som t ex att det ska gå att gå in på just den här webbsidan och sådant och det förutsätter vi att om grundfunktionaliteten funkar här, så funkar det att gå in på den sidan. Så vi testar ju inte det här nere vid *”main-branch”* utan det gör dom uppe på två.

Det kan vara så att dom börjar testa redan tidigt. Men det är ändå den organisationen, produktorganisationen som går in här. Men det sker inte på feature testningen, utan förmodligen sker det på *”main-branchen”*. Så när dom ser att nu har alla features som är relevanta blivit levererade, då börjar deras operatörstestning.

Dels kör vi regressions test uppe vid *”features”* så vi ser att ingenting har gått sönder efter att vi har pillat på det. Samma sak på maintenance, här ska vi alltid ha re-test på själva problemen och sen gör man regressions testing runt omkring för att se till så att ingenting har gått sönder.

Och sen på *”main”* har vi ju egentligen bara en enda stor regression. Alla dom 6000 testfallen är ju regressionstester.

Och sen har vi error-handling testing. Hur enheterna hanterar fel om användaren t ex trycker in något som är fel..

Vi har ju en sorts *”garbage collector”* i enheterna Typ, vad händer när den kraschar osv? Error-handling ingår liksom i våra funktionella testning. Vi har ju alltid krav på oss att testa negativa testfall. Vad händer om det här meddelandet som jag ska få har helt fel *”header”*? Kraschar telefonen då bara för att jag får en DRM-fil som har en konstig header? Nej det ska den ju så klart inte göra. Det testar vi också. All negativ testning ingår i funktionella testerna.

Två har gjort den förändringen nyligen att de börjat mer och mer med heuristic testing. Men inte vi på *”main-branchen”*. Det pratas en del om det. Vi kallar det förresten inte för heuristic testing utan för exploratory testing. Och det har diskuterats en del om det i flera år men så är alltid problemet att hur ska man få det att fungera i ett stort företag? När det måste finnas ett paper-trail och projektledaren måste kunna ta beslut baserat på informationen från testaren. Om dom inte förstår informationen så kan dom inte ta beslut. Det finns en massa problem med exploratory testing. Han, James Whitaker, släppte nyligen en bok om exploratory testing där han beskrev hur man hade använt det på Microsoft så där kan man ju se lite att här användes det praktiskt. Men det är ett problem att få det och fungera på ett stort företag. Är man bara 10 personer så är det lätt för då kan jag säga *”Kvaliteten funkar. Ja!”*, men när en rapport har gått 15 steg och upp till mig där det står *”allting ser bra ut”*, hur tog du det här beslutet? Hur ska jag kunna använda detta som underlag? Då är det lättare när man ser att av 3000 testfall så var 99% OK *”Det verkar bra, vi kör på det!”* även om det inte säger någonting mer egentligen så är det någonting som dom förstår. Plus att det kräver mer erfarna testare. T ex nu i ett annat land så har dom kanske inte samma erfarenhet och om mycket testning ska göras där så kommer dom att behöva testfallen att följa. Vi hyrde dock in konsulter tidigare som enbart körde exploratory testing, men nu finns inte dom längre här. Men dom var en unik lite

klick, dom körde nästan bara exploratory testing på hela systemet.

Så vi har kört det innan. Men det var tidigare och då jobbade vi inte alls efter den här processen. Det var innan det. Jag tror dom slutade innan sommaren.

I vilken utsträckning kan den enskilde testaren själv bestämma valet av testmetoder?

Här uppe i "features" är det mycket enskilt. Den här nya processen försöker trycka mycket på att... Det är Agile liksom, "People not processes". Vi måste använda kompetensen hos testaren och dom måste ta besluten själva för vi måste ha någon slags riskbaserad testning där vi inte exekverar... Alltså som generell antibiotika på allting för att bli av med sjukdomen liksom. När t ex dom i "features" ska testa så har dom inte så jättemycket resurser och måste testa smart och sina scope så effektivt som möjligt. Samtidigt som man då inte kan riskera kvaliteten så man inte bara säga "äh jag testat ett testfall sen skiter jag i det för vi har inte resurser" utan man måste liksom kunna säga att "Ok men om jag testat de här 15 testfallen och går lite utanför dom för att verkligen försöka hitta fel runt omkring dom också, så vet jag att funkar dom 15 så funkar 95% av grejorna". Då är det tillräckligt bra liksom. Så då gör man det valet. Men om man slaviskt följer de testfall så är dom kanske inte alls lika effektiva. Det kräver ju en testare som ägandeskap själva och som är engagerad.

Ja det finns mycket utrymme. Men allt man gör granskas sedan högre upp så man måste känna et ansvar för sin produkt och det man leverar, att det håller kvalitet. Men man får göra en "professional best guess" där liksom och gå på känsla. Ibland gör man kanske fel men om du är duktig så ska du för det mesta ha rätt liksom. Alltså du känner att du har kört tillräckligt många testfall och kan med tillräcklig högt självförtroende säga att det här fungerar. Så den här processen ger jättestor frihet till testarna att välja det dom vill. Men sen är det också problem när testare och utvecklare sitter och kör tillsammans så blir en av dom överkörd på något sätt. Att du t ex inte får den tid du vill till att göra de testerna du vill. Det kan också hända så klart. Men i teorin så finns det mycket möjligheter för att utnyttja din kompetens om du vill. Sen kan man så klart bara köra de här 100 testfallen som finns och sen behöver man inte tänka mer på det. Det är upp till var och en. Alltså... Man får ju inte göra precis som man vill, man ska ju alltid göra det optimala... Men är inte engagerad så skiter man ju i det.

Hur gick det till senast du var med och beslöt om testmetoder som ska användas?

Ska vi se här... Den gången då vi gjorde den första stora förändringen, när vi gick från... Från allra första början hade vi i princip bara funktionstest. Alltså vi testade bara funktionaliteten och det var 5 år sen ungefär. Man testade funktionaliteten och funkade den så var det bra liksom. Då funkar det tyckte man. Men sen hade vi jättestora problem med performance och kom på att det här funkar inte. Vi måste testa mer än bara funktionstest. Och då så satte dom ihop grupper som skulle analysera dom olika områdena och då var ju jag "task force leader" eller vad det nu hette för den här systemtest-gruppen. Då gick vi igenom litteratur och på nätet vilka metoder som fanns och så tittade vi på dom och sammanställde en rapport på ungefär 60 sidor. Den presenterade vi sedan för management som tyckte att det var för mycket och det måste vara mer kondenserat liksom. Vi måste ha fokusområden helt enkelt och då valde vi dom 5 (Interaction, Performance, Stability, Usability och Stress), vi och management. Så en grupp tillsattes för att titta igenom det och sen så tar management och tittar på deras material. Vi hade ju liksom skrivit om de olika metoder så då kunde management ta beslut om vilka de tyckte var viktigast.

Och vi pratade ju om hur stora vi trodde de olika områdena var... Det är ju inte lika många testfall i recovery testing som i stress testing t ex.

Så är det. Och tyckte management att vi tar bort recovery. Visst, det är fortfarande viktigt, men om vi ska trycka ut någonting till 250 testare så måste det vara mer koncist. Det måste vara mer step-by-step för alla erfarna testare dom har ju tittat på alla olika metoder och känner till dom här sakerna själva, dom behöver inget dokument som säger *"Ni borde kanske stabilitetstesta"* utan dom vet att det behövs. Men det finns många som inte har den kompetensen, som är nybörjare och som kommer direkt från universitetet eller som helt enkelt inte bryr sig. Då måste man säga till dom att ni måste testa performance och ni måste testa stabilitet. Och så gör dom det. Så det var ju en beslutsprocess. Men processen nu... Vi har satt mer ramverk för det, inte så mycket metodik. Mycket här väl kommer ifrån... Det är drivet lite från vad som projektet vill veta. Alltså... Om man tittar på systemtesten här t ex, vad vill projektet veta. Jo men dom vill ha performance trender, dom vill ha stabilitets trender, dom vill ha "user scenarios"... Dom vill ha dom här sakerna och därför förser vi dom med den informationen som dom behöver för att ta beslut. Om dom (projektkontoret) hade sagt *"Vi behöver se recovery testning"* så hade vi kört recovery testning liksom. Så på "main" så drivs det ju av vilka "stake-holders" har du och vad vill dom veta? Och det är det du förser dom med. Sen så kanske du kan hjälpa dom genom att säga *"Vill ni inte veta det här också?"* och dom tycker det är intressant. Men i "features" drivs det mer av vad behöver vi göra för att få bra kvalitet? Vilka risker ser vi? Så där är lite skillnader. Vad är syftet med testaktiviteten liksom? Är det att generera beslutsunderlag? Som det är på "main-branchen" här för projektledare och line-managers eller är det att faktiskt hitta alla relevanta fel som finns i mjukvaran?

Vilka förändringar har gjorts vad det gäller testmetoder under de senaste åren?

Intervjuperson 1: (Han ritar upp en tidslinje på tavlan) För ungefär 5-6 år sedan var det mest funktionstest men sen gick vi till funktionstest, interface test och systemtest. Alltså när vi bara hade funktionstester så var det helt enbart manuell testning. Sedan började vi fokusera dels på systemtest och dels på interfacetest... Alltså vi på main började göra det men utvecklarna var fortfarande inte riktigt med. Och sen så nu har vi gått mot förändringen... Dels då har den här TDD (Test Driven Development), börjat skriva mycket mer testfall. Sen har vi ett stort fokus på interfacetest och automatisering. Det är det som sker nu. Det är mycket krut på det. Vi ska helt enkelt automatisera. Google och Microsoft har i princip allting automatiskt, säger dom. Och då ska vi också ha det säger management. Och så är det fortfarande mycket fokus på manuell testning och på UX (User Experience).

Ja, det här har vi hållit på att arbeta fram i ett år ungefär. Nu finns det bara en utvecklingsorganisation med testare och utvecklare i samma.

Det är fokus på att... Skriver du kod så ska du även ta fram ett testfall som ska testa koden liksom. Men samtidigt så finns det kvar interface test för testare... Just lite mer övergripande så en utvecklare kanske sitter och skriver någon metod längst ner och då skriver han testfall för metoden medans vi då kanske kör en massa testfall via interfacet för att testa mer... Alla möjliga kombinationer av input och sådana grejor...

Syftet med första förändringen som jag beskrev var att man upptäckte att det

fanns performance problem och då insåg folk att onekligen har vi en stor lucka i vår testning. Vi kan inte bara testa av "Funkar mediaspelaren? Ja eller nej?", det räcker inte liksom. Det viktiga är om den klarar att spela bra liksom. I nästa förändring.. Vi hade dålig kvalitet helt enkelt. Det fanns ingen som hade ett helhetsansvar, utvecklarna ville bara leverera och skita i kvalitet och testarna stod liksom och försökte stoppa.

Och det är ingen bra setup utan nu är det liksom mer att utvecklaren och testaren sitter tillsammans. Och tillsammans ska de ta fram en bra kvalitet. Det är inte bara att leverera en kod som man tycker funkar utan man ska leverera en bra kvalitet. Så är det ju mycket Agile. Det är det som det handlar om. Allt går mot mer agile. Kraven och sådant t ex, vi hade en gigantisk kravprocess och vi måste bli mer "agile" när det gäller krav, när det gäller utvecklingen och när det gäller test. Tillsammans. Så man kan säga att det var det som drev det där i andra förändringen jag beskrev. Tidigare behövde vi bara testa enklare saker men produkten har blivit mer komplex och det sätter mer krav på testningen.

## Frågor

Hur ser utvecklingsprocessen ut på ert företag?

## Svar från intervjuobjekt 2

Vi tar fram en ny process nu som är under definition och den bryts ner i olika lager, något vi kallar lager två och de flödena som finns där är lite dimma just nu och folk arbetar från början efter ett antal principer som vi definierar så bryter man ner det till ett organisatoriskt lager ett flöde och sen bryter vi ner det. Jag kommer att förklara den processen är under utveckling och kommer att vara klar i sommar och varit på gång ett år.

Då finns det lite verifieringsprinciper i det här. Den som utvecklar någonting och levererar till någon annan har fullt verifieringsansvar på den linjen, de är också skyldiga om kvaliteten är dålig. När man levererar någonting så levererar man också testfall och testresultat, du kan inte leverera mjukvara utan att du har testresultat på det. Mellan olika leverantörer och mottagare finns det kvalitetskriterier som är godkända av båda. Det kan vara ett visst "maturity" på mjukvaran, det kan vara en viss "run time" utav fel, det kan vara alla funktioner skall vara på plats det kan vara att det finns testfall där, en checklista.

På hög nivå är det att vi har tre stycken stora distributioner, "core-plattform", mjukvara, produkt. Där emellan levererar man mjukvara. Mellan dessa tre distributioner sker integrationstest, drivare mot "core-plattform", mjukvaru-applikationer mot "core-plattform", instalerbara applikationer och kund-applikationer på plattformen. Innan dess hade vi en modell som heter "heart beat" modellen, som innebär att man har två hjärtslag per år då det kommer ut produkter en till julhandeln och en till sommaren egentligen.

Man försökte ha för att integrera risker, nya hårdvaruplattformar tog man in på sommaren för då är det mindre volymer. Då är den stabil och kommer ha bättre kvalitet när den kommer ut. Det var det man försökte och då var det en organisation som hanterade allting. Vi hade en software organisation som gjorde både "core-delen" och gjorde "bring up" på allt det där, lade på drivare och applikationer och sådär parallellt och det blev lite mycket "big bang" – integration". Det var en vattenfallsmodell, x antal månader för att specia upp "features" allting som skall in, låt oss säga att det är ett år i förväg. Sen kommer man och säger att hm det här med sociala medier är någonting som har blivit

stort, touch vill vi ha. Skall man ta in det i vattenfallsmodellen så hade det blivit touchprodukter två, tre år senare och det är inte så trevligt då. En av de sakerna man vill göra då var korta ledtiderna för att man märkte det att vi ångrar oss ganska mycket, det blir ganska mycket "waste" i och med att vi kanske skapar en himla massa "features" som inte kommer in. Säg att det kommer in tio, tjugoprocent, vi kastar väldigt mycket. I och med att vi ångrar oss väldigt sent blir det dålig kvalitet på det och då får man lägga resurser på det och då blir det ännu mindre nytutveckling hela tiden.

Vilka testmetoder använder ni er av på er avdelning och varför?

Använder ni dessa testmetoder (Lista)?  
Varför och när i processen används respektive metod?

En stor del av vår testning är att när vi har gjort klart mjukvaran går den ut för acceptanstest hos kund. De kör sina test och kommer tillbaks med feedback så rättar vi och ger dem mjukvaran igen och blir de glada då så släpper vi produkten. **Vem testar då?** Kunden testar själv med acceptanstestning så de har labbtest och så vidare, för att vi skall korta ner de här, visionen är att de skall vara nöjda första gången vi ger dem telefonen så har vi gjort avtal med de största operatörerna. Vi har deras testfall och vi ger dem en testrapport efter att vi har kört deras testfall. Det bygger förtroende och så får vi två veckors mer rättnings ledtid om vi gör detta. En av de tester vi gör som är ganska viktig för vår leverering är det vi kallar för "Internal customer acceptans" ICA. Då kör vi deras testfall och har inget val att välja testmetoder utan kör deras och det är inte alls bra för att alla de testar liknande saker.

Vi kan få en stor effektivisering om de istället ställer kriterier vad vi måste klara av eller nå för kvalitet för olika applikationer eller att vi testar av krav istället. I dagsläget har vi gjort avtal att vi testar av deras testfall. Vidare har vi för våra marknadsbolag har vi också acceptanstester för dem. Där är de dem som ställer krav och vi testar av då. Det är på samma nivå med att det är egentliga testfall i dagsläget. Det är den lilla tråkiga delen, det är validering egentligen.

Vi har andra sorters "launchkriterier" som vi bygger upp enligt ISO-kvalitetsattributen. Functionality, reliability, efficiency, usability, portability och maintainability där jag som sitter i produkt där vi egentligen inte utvecklar mer än de där kundanpassningarna som är mer datafiler xml-inställningar och bilder osv. Maintainability och portability är inte lika relevant för oss eftersom vi inte äger någon källkod. Så vi tittar mest på usability, functionality, reliability och efficiency som egentligen är kvaliteten på den slutprodukten som kunden får.

Vi har en agilare utvecklingsmetod nu. **Hur länge har ni haft det?** Några månader, säg ett halvår, och att man egentligen har tagit den här utvecklingen som egentligen var vattenfallsmodell innan och kapat ihop den till tre stycken parallella flöden och kapat ledtiden med x antal månader så det betyder att när vi testar och vi har kommit in och testat tidigare när vi har haft "core complete" och gjort testning fram till beta. Det betyder att vi måste ändra vårt tänk ganska mycket, förut har vi haft stora block systemtester som egentligen är kanske tiotusen testfall mot att nu köra minimalt med testning vecka efter vecka och välja lite smartare "test scope". Då har vi det här integration då när man lägger på en kundanpassning på en applikationsmjukvara som t ex byter nått långt ner och de kan gå riktigt långt ner de här kundanpassningarna. Något som vi också kör mycket nu är heuristic based testing, eller exploratory testing vilket vi har gjort i 1-2 år nu. Strävar ni efter att automatisera så mycket som möjligt? Ja, speciellt när man tittar på att vi har tagit en process som var vattenfallsmodell som har mycket kostnader på testning sent och splittat upp det till en parallell process kan det hända att ett och samma krav kan komma att testas tre gånger,

finns det en integrationsrisk lägger man på en applikation eller en server som ligger någonstans så kan det hända att man behöver testa om någonting i core-plattformen som utvecklades av software sen kanske i produkt igen för att man lägger på en kundanpassning. Det är dyrare att ha tre parallella flöden samtidigt testmässigt än att ha ett. Då har man helt plötsligt en annan return of investment på att automatisera än man hade tidigare. Vi talar också om att vi har en "continuous development" förut hade produkten en start och ett slut, man branshade ut sen levde produkten ett tag sen döda man den vilket betyder att alla rättningar och så måste man göra på två ställen. Nu har vi ett main-spår som påbörjas och aldrig slutar egentligen vilket betyder att de testfallen som man gör på core-plattformen kan man återanvända om de är automatiserade på mjukvara och sen på produkt så du får en tredubbel return of investment direkt bara du kör det en gång.

Vi har nått annat som är ändrat i den nya processen som är att kravorganisationen specificerar mer högnivå krav och är en stakeholder och produktägare när man har SCRUMliknande utveckling.

Alla testmetoder som ni har på er lista använder vi oss av, förutom error-handling testing som vi använder, men inte särskilt mycket. Det finns inte tillräckliga syften för att utföra det mer än vad vi gör. Alla testare får ansvar för en liten bit, men inte för mycket. Sen är det egentligen inga tester som sker tidigare eller senare utan allting testas från början till slut i princip. Det har blivit så nu när vi börjat jobba mer agilt.

I vilken utsträckning kan den enskilde testaren själv bestämma valet av testmetoder?

Regressionstestning styrs av processen och där kan de inte välja någonting. När det gäller att automatisera så sitter det på testobjektägaren att driva en sådan road map. Det här testfallet kör vi tiotusen gånger så vi har en return of investment på att köra det och det är upp till ett enskilt team att välja om de skall automatisera något eller inte.

De tar också fram testfallen, det är testarna som tar fram verification intent och säger det här behöver vi stress-testa, det här behöver vi performance-testa sen är det jag som godkänner om det är tillräckligt så jag är accountable om någon har gjort ett fel val, men de är testarna som tar fram det.

Hur gick det till senast du var med och beslöt om testmetoder som ska användas?

Jag är med bland dem som designar processen och hela det nya då och där arbetar vi i en grupp där vi har en som leder forumet så är vi kanske fyra personer, en processansvarig, en arkitekturansvarig, en projektansvarig och kanske en senior chef. Vi sitter och arbetar fram processen, vi tittar på vad som var definierat innan, så här kommer det att ske, det var nått de gav till oss, vi kommer att leverera varje vecka nu. Hur skall vi hantera det nu? Skall vi bunkra ihop testningen till var annan vecka eller skall vi testa varje vecka, var femte vecka, skall vi testa på varje mjukvara eller var femte, var tionde? Sen har vi den här inputen som de gav till oss alltså de som levererar någonting måste ha testat av det fullt ut.

Vi tar fram ett förslag på hur det skall fungera och sen godkänns det av en styrgrupp som är högsta chefen för test.

Det är för att vi håller på att ta fram en ny process så det är processtyrning på högsta nivå, alla skall byta process. Det är lite specialfall. I framtiden kan det komma från den här enterprise-arkitekten jag berättade om innan, det är han som skapar styrdokumentet för hur man skapar testfall. Det är egentligen bara definierat så att det finns ett styrdokument, så här skapar du testfall. Innehållet i den är inte definierad, det finns inget sådant dokument idag för det är någonting

som han kan styra ner. Han är ansvarig för all testning så om vi missar någonting och en kund hittar det så får han skulden för det. Han kommer att ha ett styrdokument som säger hur mycket testtäckning vi skall ha. Man kan inte säga "gör om, du gjorde fel", utan man måste ha gett dem chansen att göra rätt från början. Det finns ett styrdokument som heter verification design scope guide men innehållet i det finns inte.

Vilka förändringar har gjorts vad det gäller testmetoder under de senaste åren?

Det är ganska mycket, om man tittar på mjukvara då så har det gått från att bara scriptade testfall mycket funktionellt. Vi har haft en viss stabilitets-test vi har haft en viss sorts performance-test som är drivet av krav. Kravavdelningen har fått definiera upp min och maxvärden och så har vi antingen ett pass eller fail alltså svart eller vitt till att kravorganisationen definierar högnivå krav och så bryter vi ner dem till nått som är applicerbart. Det finns för och nackdelar med det. Fördelarna är att man kan testa av det som är relevant eller det som verkligen behövs. Nackdelarna är om man skall lägga en felrapport på någon att performance inte är tillräckligt bra så måste man ha ett godkännande av den felrapporten. De som äger koden hade ingen aning om hur snabbt det skulle vara och om vi säger att det är för långsamt så är det lite taskigt, eller att det drar för mycket ström. Finns det inga sådana definitioner så måste man ha en mjuk godkännandeprocess, det måste sitta ett triage mellan alla organisationer och säga det här skall vi rätta det här skall vi inte rätta och göra de valen. Eftersom man menar att det är kvalitet så kan man visualisera att t ex den är långsam och så får man ta beslut efter det t ex att fokusera på performance. KPI:erna som visar upp hur snabb performance är godkänns på högsta nivå så om vi skall mäta strömförbrukningen på de tjugo vanligaste applikationerna så får vi in kunddata på vilka de tjugo vanligaste applikationerna är, så har vi en godkänd KPI på hög nivå som säger att då kan vi lägga fel på det om den inte klarar av det. Vi klarade inte av det, vi hade femhundra personer som jobbade med krav innan men det var inte alls tillräckligt bra eller tillräckligt många krav. Nu testar vi det som är relevant istället, tittar på vad operatörerna vill ha, vad slutkunderna vill ha. Om man använder en riskbaserad testning här när man väljer ut vad man skall köra vecka efter vecka så om det är en liten applikation som kanske bara finns på en regional marknad eller som inte används så mycket så testar man inte den utan lägger testning på samtal som används mycket mer eller att den inte drar för mycket ström när man spelar musik. Varje avdelning skall testa sitt eller förlåt, jag sa fel, har ansvar för att testa sitt. Ingen behöver testa nått men de har ansvar för det.

### Frågor

Hur ser utvecklingsprocessen ut på ert företag?

### Svar från intervjuobjekt 3

Så om jag beskriver den på en väldigt hög nivå... Man har brutit upp utvecklingsmodellen i tre parallella huvudprocesser. En Core-plattformprocess, en softwareprocess och en produktprocess. Core-plattform är kontinuerlig utveckling av plattformsteknologi, d.v.s. att du tar hårdvara och chipset och en core-mjukvara externt, så tar du in dom och gör dom modifieringar vi vill göra för att kunna göra stommen i våra produkter. Så den är kontinuerlig, den fortsätter hela tiden så det är ingen start-stopp process. Däremot så tar du ju in nya plattformar som en del där, men processen i sig är kontinuerlig. Ovanpå den sen så har du mjukvaruprocessen, så baserat på att det hela tiden finns



en core-plattform, senaste varianten, så lägger vi på mjukvara. Eller vi vidareutvecklare den mjukvara som vi har. Och det är samma sak där, det är en kontinuerlig process. Så du har alltid senaste mjukvara och så lägger du bara till nya features på den som du utvecklar efter hand och säkerhetsställer att kod-basen alltid är stabil. Och sedan, det tredje steget är i princip när som helst så kan du droppa mjukvara från den här gemensamma kod-basen till produkterna. Eller snarare, du utvecklar... Stödet för en produkt, det är en feature så nu ska vi stötta en ny produkt. Då tar du den produktkonfigurationen av kod-basen, det är en feature som du tar in. Och från den dagen du tar in den featuren, så kan du stötta en ny produkt. Och då kan du starta produktprojektet, som till stor del då är mekanik- och industrialiseringsprojekt. Så ser den nya utvecklingsmetoden ut.

Använder ni er av agila utvecklingsmetoder? Om ja, på vilket sätt och hur påverkar det testningen?

Ja, definitivt. I den här nya utvecklingsmetodiken som tas fram... Hela den metodiken, kontinuerlig utveckling, är ju i princip och i grunden agil. Det påverkar testning som så att man... Dels går man mer åt mycket mer skriptad (automatiserad) testning jämfört med tidigare där vi har haft rätt så mycket manuell testning. Det är väl en av dom stora skillnaderna skulle jag vilja säga. Och du automatiserar mer för att... I vattenfallsutveckling så så är det ofta så att man låter kvaliteten sjunka under alpha och beta och så jobbar man för bättre efter det. I det agila så vill du hela tiden ligga på en stabil kvalitetsnivå på det du tar in, vilket gör att du kan inte ha för långa perioder för regressionstestning av kod-basen så regressionstestningen måste... Tiden för den måste kortas ner. För annars kan du inte vara agil. Har du för lång regressionstid så hinner kod-basen bli dålig hittar. Och då faller hela systemet. Så det är dom två stora skillnaderna. Dels måste du göra mer automatiserad testning som ligger närmare utvecklingen och så måste din regressionstid minimeras. För att kunna säkerhetsställa utvecklingen så att säga.

Använder ni dessa testmetoder (Lista)?

**Stress test?** Jepp. I korthet så... För att få en heltäckande bild av kvaliteten av produkten så måste du egentligen täcka alla dom olika områdena. Man kan referera till kvalitets-attributen. Du vet att du har bra täckning om du testfall som testar alla attributen. Så för att testa reliability så måste du stress testa. ISO-9126, det är kvalitets-attributen. För att testa usability så måste du ha en viss typ av testmetodik, för att testa reliability måste du ha en viss testmetodik, för att testa efficiency så måste du.. Ja ni förstår. Och vi har i dagsläget inte formell spårbarhet mot dom attributen, men vi har täckning som täcker i princip alla. Så att vi har stress tester, vi har duration tester, vi har performance tester, vi har usability tester, vi har rena funktionella tester, vi har conformance tester. **Recovery test?** Japp. **Requirements test?** Japp. **Error-handling test?** Det finns inte en formell metodik som spårbarhet med vilken täckning vi har error-handling, men fel-falls metodik har vi definitivt. **Heuristic testing?** Jepp. Vi började formellt att tillämpa heuristic testing för ungefär 2 år sedan. I alla testområden så tillämpas heuristic testing, ungefär 10-20% av resurserna.

**När i utvecklingsprocessen används respektive metod?** Vi använder alla från första dagen vi kommer in i projektet. Om man tittar enligt den nya utvecklingsmodellen så ska alla passera alltid. Så i den nya modellen så är det inget som är lite viktigare en viss tid än en annan. Tittar du historiskt sätt så har det ju varit så att, om vi tittar på projekt som fortfarande ligger kvar i gamla

	<p>processer, då är det ju vattenfalls-baserat och då är ju givetvis rena funktionella tester det viktigaste. Inte viktigaste, men dom är att köra först och när du har det funktionella på plats så kan du gå in på tillförlitlighetstester, stabilitets- och felhanteringstester. Och sedan när du har dom på plats och du börjar närma dig produktion så kommer man in på security, d.v.s att säkerhetslösningar i produkten. I övrigt så kan man säga att vi tillämpar alla testmetoder parallellt i den utsträckning som tillåts.</p>
<p>Finns det testmetoder som tidigare har använts men inte längre? Om ja, varför?</p>	<p>Nej dom går igen... Vissa är svåra att döda. Nej, men det växlar väldigt upp och ner. Utvecklingsmetod och organisationen förändras, så den organisationen jag sitter i just nu har ändrats väldigt mycket under fem år. Men tittar jag på hela utvecklingskedjan så är det egentligen inte någon jättestor förändring. Så nej, det är inte någon testmetodik som vi slutat att använda.</p>
<p>Känner du till testmetoder som kan vara relevanta för er men som inte används? Om ja, varför?</p>	<p>Inte testmetoderna i sig, men man kan ju alltid utveckla sin mognadsgrad i test. Men jag kan inte säga att det är... "Här har vi ingen testning" eller "den här testmetodiken saknar vi", det tror jag inte. Det är jag rätt övertygad om... Vi har alla testmetoder, utan det är mer mognadsgraden i utvecklingskedjan som sådant som spökar för oss. Hade vi saknat en testmetod som behövs, så får man den i huvudet när produkten går ut på marknaden. Och med tanke på att vi lanserar så många produkter så lär man sig.</p>
<p>Vilka förändringar har gjorts gällande testmetoder under de senaste åren? Vad var syftet och vad uppnåddes?</p>	<p>För fyra år sen så skiljde vi på plattform- och produkttest, vilket vi då kallade för funktions- och systemtest. Funktions- och systemtest är ju egentligen två metodiker som kan tillämpas parallellt, men vi delade upp dom organisatoriskt. Det var när vi jobbade väldigt hårt vattenfallsmetodmässigt. Sedan så... Men testmetoderna som sådana, alltså vilka typer av testfall som ligger där dom förändras inte. Ja automatiseringsgraden och möjligheten att jobba med... Förutsättningarna för Return on Investment på automatiserad testning förändras när man går in i den agila arbetsmiljön. Så i början så var det mycket automatiserade tester, men sedan började produkten innehålla mer och mer applikationer och då ökade möjligheterna för mer UI-driven testning. Och sedan så när det började bli stort så fick du igen, för att dra ner kostnaderna, automatisera UI-driven testning. Och nu går vi in i dom öppna systemen och agil testmetodik igen så har vi ny utvecklingsmiljö och mycket större möjligheter att inför sådana här automatiserade regressionstester. Så då går vi tillbaka till skriptade tester igen, fast på applikations-nivå. Man strävar alltid efter att automatisera så mycket som är ekonomiskt försvarbart. Och det är ofta det som är kärnan. "Hur ser ens utvecklingsprocess ut?", så får man göra en "business-case analys". Hur många gånger behöver vi köra detta testet, hur lång tid tar det att köra detta testet, vad kostar det att köra denna testningen manuellt? Ställ det mot: Hur lång tid tar det att automatisera den testningen, hur lång tid tar det att köra den automatiserade testningen och hur mycket kostar det att underhålla det automatiserade testet? Så får du totalkostnader för båda scenariona och så ser du vilket som är billigast. Så väldigt ofta, om man tittar på vår typ av industri med väldigt korta projekt-tider och stor förändring t ex, så är det kanske inte alltid ekonomiskt försvarbart att automatisera.</p>
<p>I vilken utsträckning kan den enskilde testaren själv</p>	<p>Troligtvis större del än vad som borde vara bra. Det är en väldigt stor organisation, och i en stor organisation så behöver många små bitar passa</p>

bestämna valet av testmetoder?

ihop. I en sådan miljö så måste, för att alla kuggar ska passa in perfekt, så måste det vara väldigt tydligt vad som förväntas från de olika stegen i utvecklingen. Så det ska finnas guidelines egentligen för varenda enskilda testare att följa. Sedan beror det på vilket område han sitter i, om han får välja black box, white box testing vilka testverktyg han får och får inte välja. Så det är lite svårt att svara generellt på en sådan fråga utan att peka på en viss del av organisationen. Det finns oftast väldigt naturliga val, beroende på vad ditt syfte med testet man ska göra är så är det oftast väldigt naturligt att jag ska välja just den här testmetodiken eller så. Det styrs nästan av var man sitter. Inte så att man tvingar testaren eller så utan för att han ska kunna lösa sin uppgift så finns det få att välja på.

Kan du beskriva hur det gick till senast du var med och beslöt om vilken testmetod som skulle användas?

Mmm... Det gör jag fortfarande rätt ofta, även om det inte är på så praktisk nivå längre. Eller jo till viss del. Det är alltid samma struktur. Man tittar på de olika alternativ som finns, man tittar på vad "impacten" är av dom. Pros and cons eller business-case beroende på hur situationen är. Jämfört med syftet man ska uppfylla. Man gör helt enkelt en analys av vilka möjligheter som finns. Och så presenteras beslutsunderlagen, om vi tittar på lite högre nivå och inte rent operativt, "Vad behöver vi göra för att uppnå målet?" sen presenteras man de olika alternativen. Ett praktiskt exempel: Vi har ett projekt som vi tittar in på förbättring av stabilitetsområdet just nu. "Dom här testverktygen kan vi använda, vi kan använda de här metoderna för att skapa test-täckning och vi rekommenderar de här och de här." Så får projektet presentera x antal alternativ, oftast tre stycken, och beslutar styrgruppen att man väljer den under de här förutsättningarna. Tittar man sen på lägre nivå, den enskilda testaren som sitter och arbetar, så har han ett testcase designguide som styrdokument som han får från sin testarkitekt som säger "Du ska designa testfall för ditt område enligt den här metodiken." Så är det styrt på den nivån. Så arkitekten har valt hur man ska testa de olika delarna. Komplexiteten i organisationen påverkar hur mycket man själv kan besluta och hur lite toppstyrda besluten är ju komplexare organisation desto toppstyrdare. Organisationen är väldigt stor och när jag började 2001 var det väldigt toppstyrt men har gått mot en mer lös beslutsordning till att idag vara tillbaks vid en toppstyrd beslutsprocess.

Projektens storlek påverkar också hur beslutsprocessen ser ut. I mindre projekt är man friare att ta egna beslut medan i större projekt är de flesta besluten redan tagna.

Heuristic testing betyder inte att testaren kan besluta mer själv utan det beror på var i utvecklingsprocessen man befinner sig. Tidigt i utvecklingsprocessen är det mer bestämt hur man skall testa men ju närmare man befinner sig release desto friare blir testaren i sina beslut.

## Frågor

Hur ser utvecklingsprocessen ut på ert företag?

## Svar från intervjuobjekt 4

Nej, jag bara funderar på hur man ska, ja alltså, man kan väl beskriva det ... isolerad, eller va ska man säga... med stora penseldrag. Först har man en kravanalysfas såklart, vad är det för en slags produkt vi ska göra, om vi börjar

i den änden. Man beslutar att man ska göra produkten därefter, när man vet att man har resurser till det här. Därefter så börjar man rent mjukvarumässigt gör man featureutveckling isolerat för att hitta problemen isolerade så att de inte stör resten av utvecklingen, kan man uttrycka det, men därefter när de här är tillräckligt bra så stoppar man in de i produktmjukvaran, eller vad ska man säga, i mainbranchen som vi kallar de då, som alla har tillgång till då och testar av hur funkade det här i systemet då. Tanken är egentligen att man ska testa de här featurena systemmässigt redan under featureutvecklingen också, det görs i viss utsträckning ska vi säga, men det blir ändå implementationsproblem som man ser först när man har fått in det tillsammans med annat också. Och därefter när allting då är fixat tillräckligt bra, så går vi ut till kund-test, det vi kallar för CA inom företaget, Customer Acceptans, där kunderna då, leverantörerna, utför sin test i sina nät ur sin synvinkel. Vi gör förtester inför det här så att vi givetvis ska kunna få en lätt resa när det här väl utförs hos kunderna så att inte de ska bli missnöjda. Och därefter så fixar vi det sista för sen är det lansering.

Använder ni er av agila utvecklingsmetoder? Om ja, på vilket sätt och hur påverkar det testningen?

Agila metoder använder vi oss av, framförallt i de här feature-teamen då, alltså om vi pratar Agil/SCRUM så, ja och hur de används det är väl egentligen för att få, ska vi se här, om man staplar upp alla kraven vi har till exempel om man ska ta fram en produkt så har man ju "core", eller vad ska man säga, Key-features som verkligen är ett måste för den här produkten. Och man har features som är väldigt komplicerade att göra, den här kommer först i kedjan, och sen så då har vi ju ett pärlband av lägre prioriterade "nice to have"-features o.s.v. och det, hur man ska säga, hur det här kommer in med det Agila det är väl i så fall att man bryter ner allting i dom här olika featurerna då isolerat, att man ska skulle kunna säga att man tar fram en produkt, har vi kommit halvvägs, tiden är slut, ja men ok, är det bara "nice to have" kvar? Ja men vad bra, då skiter vi i dom och så lanserar vi nu, om du förstår vad jag menar. Så att, det för mig är min tolkning av vad Agil utveckling är. Ja, och det gör vi ju för då kan vi ju testa av dom här del-leveranserna på ett mer kontrollerat sätt än om man har väldigt många parallella spår som går som sen slås ihopa i slutet så att säga, så att det gör att får man in nåt som är dåligt kan man slänga ut det ur produkten utan att det skadar övrig test och utveckling. Så att den används för att hålla en stabil test- och utvecklingsmodell skulle jag nog kunna säga.

Använder ni dessa testmetoder (Lista)?

Ja vi använder alla de testmetoder som ni har på er lista och listan kan göras mycket, mycket längre. Alla testmetoderna används i nästan direkt från början av projektet. Alltså egentligen körs många av testerna på en feature-nivå, där man då sitter, där man gör sin feature, dom, ja dom säger för att leverera in på det här main-spåret, s.k. huvudspåret, så har vi integrationskriterier och dom, där är ju detta delar av det så att säga stress-test, du ska tåla så här mycket stress-test, fast då kallar vi det något annat då så att säga, och kod... eller vad ska man säga, kod-analys gör vi också med statiska verktyg som går in och tittar på koden, finns det något..., är det något som är felkodat och du inte hittar det med dynamisk testning... du kanske inte ser det med dynamisk testning men du ser det med statiska verktyg. Så att alla dom här täcks ju upp så att säga för att man ska få leverera in någonting, så att allting börjar redan där. Useability börjar till och med innan, även om den inte fanns med på listan, den börjar i designfasen där man faktiskt rent krasst sätter upp

gula lappar på en tavla i nåt scenario bara för att testa flödena eller verkar det här logiskt, så till viss del gör man ju en del testning innan man implementerar saker och ting till och med, eller innan man ens skriver en kodrad. Inte stresstest såklart men... vissa saker är helt automatiska, randomtest, alltså man kan säga att den automatiska stresstesten är dels en randommiljö och dels en statisk miljö där du går igenom allting precis på samma sätt, och sen det tredje fallet är ju i kritiska områden, vi har en ny feature, då stresstestar vi den för att veta att ja, den måste ju vara 100%. Eller där vi ser mycket fel, man kan säga så att det är en riskbedömd insats där vi väljer att göra stresstest, så man kan säga att det är alla dom här tre, dels slumpartat, dels statistiskt på hela systemet och dels punktinsatser där det behövs, vilket då utförs i princip i hela utvecklingskedjan. Både på featurnivå och på systemnivå.

Recoverytest; Den har jag svårt att sätta fingret på hur, jag har inte upplevelsen att det görs jättemycket i den definitionen så att säga här, så jag har svårt att säga när och var den görs i så fall

Requirements...

Nu för tiden är det till viss del också innan man har skrivit en kodrad så förväntas man skriva en testrad faktiskt, och sen givetvis hela vägen igenom men kanske med mindre omfattning i den senare delen om man nu säger att "har vi autozoom på kameran", vet vi det tidigt så är det rätt så osannolikt att denna feature försvinner, så att det har väl kanske mer tyngdpunkt i den tidigare fasen.

Det är det som balanserar om man tänker på requirements-testerna som en funktionell test sen är det ju nästan samma saker vi gör fast då kallar vi det regressionstest. Har det gått sönder eller inte, är ju det svaret man vill veta när man har gjort ett regressionstest.

Om vi översätter till negativ testning så skulle jag nog vilja säga att den också är i den tidigare testfasen där vi då sen kör regressionstesten över de negativa testerna där Error handling handlar om att utmana systemet medan du har tid att fixa det, om du lägger den utmaningen sent då har du inte tid att parera om du skulle hitta något skit. Så generellt all testning försöker du trycka så tidigt det bara går, redan på tankestadiet eller i nåt fall innan man gör koden eller direkt efter att koden är klar.

Finns det testmetoder som tidigare har använts men inte längre? Om ja, varför?

Nej, det tror jag inte.

Känner du till testmetoder som kan vara relevanta för er men som inte används? Om ja, varför?

Nej, jag tror inte, eller jo, kanske typ som recovery till exempel, där tror jag spontant att vi är lite svaga, det kan vara så att det är ok att vi är lite svaga där så kan det finnas områden inom det här som man skulle kunna hitta mer om man gav det mer tid, eller i alla fall det ger svaret att är det ett medvetet avsteg eller är det tidsslump och resurser som har styrt valet om du förstår vad jag menar. Just recovery tror jag, om vi nu är svaga på det så är det mer tidsslump och kunskap som styr att vi kanske gör det i mindre omfattningar

Vilka förändringar har gjorts gällande testmetoder under

Jag skulle nog vilja säga att vi, dels trycker vi testningen tidigare i projekten, dels är det att vi vill att på funktionell utveckling ska man tänka

de senaste åren? Vad var syftet och vad uppnåddes?  
systemmässigt. **Hur menar du då, systemmässigt?** När man utvecklar en feature är det lätt hänt att man zoomar in på sin feature isolerat, hur funkar den, men du tänker inte hur den interagerar med, om världsfunktioner eller hur de interagerar med din feature. Det är väl nånting vi försöker trycka så att man tänker system redan från början, alltså även på featurenivå istället för att det upptäcks när det väl är integrerat. Det är väl en sak som är ganska ny. Heuristictestning är väl ett ganska nytt, det är nytt i det sättet att, vi har alltid gjort det, men Heuristic sätter ett namn på det så att vi kan göra det på ett strukturerat sätt. Som det var tidigare bygger det givetvis på att du har erfarna testare som gör Heuristic testing, fast de vet inte att det är Heuristic testning de gör, om du förstår vad jag menar, så att egentligen är det inte något nytt, egentligen har vi satt ett namn på något som vi har gjort innan och sedan sätter vi en struktur på det dessutom. I övrigt har jag nog svårt att säga några andra saker som vi har lagt till som sagt de senaste åren, tittar man riktigt långt tillbaks igen så var fokus mer på funktionellt test än på system test, men då är det långt tid tillbaka. Syftet med detta är att vi vill undvika "överraskningar" senare i projektet att saker inte passar osv.

I vilken utsträckning kan den enskilde testaren själv bestämma valet av testmetoder?  
Begränsad skulle jag vilja säga av valet, utan det styrs nog mer av dom som styr testningen och kvalitetspersoner eller utvecklingstesten är nog mer utförandedelen så att säga men givetvis i en heuristictestning där förväntas han att vara, tänka mycket mer själv.

Kan du beskriva hur det gick till senast du var med och beslöt om vilken testmetod som skulle användas?  
Ja,... bra fråga, Jag skulle nog vilja säga någon månad sedan och i sammanhanget att vi kontinuerligt förfinar vår utvecklingsmodell och därav har diskussioner inte bara om testmetoderna i sig utan det finns en variabel till som spelar in och det är testmiljön och då menar jag inte bara att du har den på kontoret eller utanför kontoret, inte sådan fysisk miljö utan mjukvarumiljön, är det en produkt som är kundanpassad eller inte kundanpassad, vilka typer av testfall ska utföras på de här två olika varianterna. Vi vill ju..., så då ledde jag det teamet på x som satt egentligen och ritade upp kartan för hur ska x organisation jobba framåt, vilka..., det var kanske inte så att vi diskuterade vilka testmetoder ska vi ha och vilka ska vi inte ha, men indirekt så gör vi ju det. Man kan nog säga att, man zoomar liksom bakåt och funderar på att ok, nu måste vi lägga hela bilden, vi tittar på vad gör x som organisation, vad gör mjukvara som organisation, vad gör en annan organisation som heter y. Vi måste fylla hela pusslet, vem gör vad, vem förväntas göra vad rättare sagt, och då, det är då man kommer in på det här med, kanske inte testmetoder, mer testansvar inom de olika organisationerna vilket indirekt är kopplat till testmetoder också, så att säga.

## Frågor

Hur ser utvecklingsprocessen ut på ert företag?

## Svar från intervjuobjekt 5

Det beror dock lite hur ni menar och vilken plattform ni menar... Var i livscykeln och så. Men det är väldigt stor skillnad nu jämfört med några år sedan och det gamla sättet. Vi är på väg in i det vi kallar för WCDM (World

Class Development Model). Och den har också i sin tur ändrats 2-3 gånger sen vi började med den. Jag vet inte riktigt hur mycket detaljer ni vill att jag ska gå in på? Den gamla modellen är mer eller mindre död. Den är död, kan man säga, även om bitar av den sprattlar lite fortfarande. Men i praktiken är den död. Och så går vi då in i en mer agil modell.

Använder ni er av agila utvecklingsmetoder? Om ja, på vilket sätt och hur påverkar det testningen?

Väldigt mycket. Testare och utvecklare mixas mer upp och sitter i samma team. Vi skakar om begreppen om vad som är utvecklare och vad som är testare. Vi kommer också att få en annorlunda relation till våra vendors. Så beroende lite på vilket chipset dom har och vilken del av deras kod som vi vill behålla så måste vi se vad vi behöver testa när vi tar emot från dom. Vi kommer att testa den del för våra vendors då vi har en närmre relation till dom än de flesta andra.

Använder ni dessa testmetoder (Lista)? När i utvecklingsprocessen görs respektive metod?

Ja allihop. Vi gör dom definitivt på alla nivåerna, det gör vi nog. Jag skulle dock vilja trycka ner dom ännu längre. Men vi jobbar med dom här på alla nivåer så att säga, det är bara scopen och perspektiven som är olika. **Så det är ingenting som görs bara i början av utvecklingsprocessen t ex?** Nej... Tyvärr är där en del människor som inte greppar det här. Men alla olika attributen som testas, kan man göra på alla nivåer... Och även i tid. Det är bara det att perspektivet är annorlunda. Så du kan göra en stress test på en liten komponent, och det gör du ju väldigt tidigt. Och då har du "den" synen på det. Sen gör man ett stress test på ett helt system, och det gör man ju mycket mycket senare. Då har du ett annat perspektiv och ett annat syfte. Så därför kan man inte säga.. När, i tid, man gör det utan det är perspektivet. Men givetvis så brukar vi börja med funktionella tester.

Finns det testmetoder som tidigare har använts men inte längre? Om ja, varför?

Det är en bra fråga... (Tänker)... Det skulle vara att vi har gjort en hel del manuella saker som vi vill komma ifrån. **Som ni vill automatisera då eller?** Precis. Så att vi kommer lämna sådana tester bakom oss. Och automatisera mer och mer. Det är mer och mer rent tekniska tester också nu. Men med detta sagt så tror jag fortfarande manuella tester behövs. För det är ju då man hittar felen. Det är en enorm övertro på automatiserade tester, och så är det överallt inte bara här. Automatiserade tester är inte alltid det bästa sättet att hitta fel på. En mänsklig hjärna är... **Men det är kanske mer ekonomiskt försvarbart ibland?** Ja, det är det. Och just för upprepadet och så...I min värld så är automatiserade tester oslagbara när det gäller att skydda sig mot regressioner. Och när man måste ha stora mängder data eller tider och så där. Men har man ett UI och det gäller att hitta nya fel så är den mänskliga hjärnan totalt överlägsen. Det är min synpunkt.

Känner du till testmetoder som kan vara relevanta för er men som inte används? Om ja, varför?

(Tänker)... Mycket av det jag har pratat om hittills och det vi vill göra... Vi är inte riktigt där än, men jobbar på det. Så att ja, jag känner till metoder som vi borde göra men vi har inte kommit dit än.

Vilka förändringar har gjorts gällande testmetoder under de senaste åren? Vad var syftet och vad uppnåddes?

Syftet är att vi ska öka kvaliteten på produkterna. Vi ska korta deltid och göra saker och ting billigare. Alltså grundsyftet med alla företag är att man ska tjäna mer pengar. Det är därför vi gör det (Skratt). Sen är det ju så, handen på hjärtat, har vi levererat med för låg kvalitet. Så är det. Och vi ska

tillbaka till att ha den fina kvalitet vi hade för några år sedan då vi faktiskt var ett av dom företag i världen som hade bäst kvalitet.

I vilken utsträckning kan den enskilde testaren själv bestämma valet av testmetoder?

Den enskilde testaren har alltid väldigt stort utrymme och ansvar på sitt område, men där är alltid en arkitekt ovanför som ska godkänna det. Så dialogen där... Det är också nånting vi inför. Men alla, tom den enskilde testaren har ju sitt område. Och inom det området så finns det stora möjligheter, man uppmuntrar och vill, att man ska föreslå förbättringar. Sen är ju inte alla av samma kaliber. I alla fall dom som jobbar för mig, så är det så. Vi har tom sådana här utvecklingsmål. Alla har sitt eget område och så har de ett mål för året där de ska utveckla sitt område ur som här 6 kvalitetsattributen (ISO-9126). Man tar ägandeskap av sitt område.

Kan du beskriva hur det gick till senast du var med och beslöt om vilken testmetod som skulle användas?

Det var väldigt lyckade resultat när jag införde det på förr avdelningen så denna gången var det inget större beslut utan det var bara att köra. **Så hur gick det till första gången?** Då var det lite segare (Skratt). Första gången.. Jag började jobba med exploratory testing 2001. Sen har jag samlat på mig en massa erfarenheter efter hand. Jag fick helt enkelt börja utbilda folk på min förra avdelning om vad det handlade om. Och efter ett tag så växer det.

## Frågor

## Svar från testare 1

Hur länge har du varit på företaget?

I ungefär 2 år

Hur länge har du jobbat med testning?

Lika länge, ca 2 år

Kan du själv välja hur du skall testa, om ja beskriv när och hur?

Ja, alltså det är väl både och. Det finns ju alltid specifika saker som måste bli testade helt enkelt så på så vis är ju friheten inte så stor. I många fall är det ju också steg-för-steg tester som vi utför, men det är ändå inte så att man helt strikt måste göra så och så utan man är uppmuntrad till att "experimentera". Improvisation är ett måste ofta. Sen så har vi ju sådan här heuristic testing, eller exploratory testing, där man testat helt fritt vilket område som man känner för en viss tid.

Vad baserar du dina beslut på när du bestämmer vad som skall testas?

Nästan alltid vet man ju vad som ska och behövs testas, det har gjorts så många gånger och man vet vad som behövs. Men t ex när man kör heuristic testing så kanske man märker att en viss feature inte helt fungerar som den ska och då tar man ju beslutet själv att testa den lite extra kanske. Detta uppmuntras så klart och det är bra.

Hur mycket i procent skulle du säga att du bestämmer och hur mycket är bestämt?

Svårt att säga då det varierar... men jag skulle nog vilja säga att det är 40% jag och 60% är redan bestämt. Kanske lite mer är bestämt redan, men det varierar som sagt. Det skulle inte funka om man hade total frihet som testare.

Skulle du kunna bestämma

Bra fråga. Ibland kan jag tänka mig att det är så. Det är ju så klart "bekvämt" ju mindre



mer själv men väljer du att inte göra det?

beslut man måste ta, men jag försöker att alltid bestämma själv och improvisera när det är möjligt och tillåtet.

Har det att göra med hur rutinerad man är på hur mycket man som testare beslutar själv?

Det tror jag absolut och då talar jag från min erfarenhet. Man är så klart lite osäker i början och då är det ju lättast att följa de andra och så. Men sen blir man mer rutinerad, och man lär sig hur allting funkar... Då tar man fler beslut själv helt klart.

## Frågor

## Svar från testare 2

Hur länge har du varit på företaget?

Sedan 2005... Så 5 år.

Hur länge har du jobbat med testning?

Har jobbat med testning sedan jag började, med ökande ansvar genom åren. Är testledare nu vilket i princip är samma sak som projektledare.

Kan du själv välja hur du skall testa, om ja beskriv när och hur?

Ja till viss del kan man välja vad man ska testa. T ex så har vi ju sådan här testfalls-testning där redan färdiga steg-för-steg testfall ska göras, och på det sättet kan man inte "välja" vad som ska testas. Valet ligger i att du kan välja vilket område du vill testa och kanske t.o.m. specialisera sig på. Så du väljer vilka testfall du vill ta dig an. Sen har vi ju heuristic testing som även kallas för fri-testning där man helt enkelt får välja helt själv vad man vill testa.

Vad baserar du dina beslut på när du bestämmer vad som skall testas?

Jag baserar det bl a på tidigare erfarenheter, man vet så att säga i vilka områden som det kanske brukar vara problem. Sen kan det vara så att man sitter och testar en ny mjukvara och märker att något specifikt i mjukvaran strular och bestämmer sig då för att testa det området extra mycket.

Hur mycket i procent skulle du säga att du bestämmer och hur mycket är bestämt?

Skulle gissa på att det är 20-30% fritt

Skulle du kunna bestämma mer själv men väljer du att inte göra det?

Kan jag bestämma själv så gör jag alltid det. Jag tycker det är viktigt att ta egna initiativ men självklart måste det alltid finnas gränser.

Har det att göra med hur rutinerad man är på hur mycket man som testare beslutar själv?

Ja det är jag helt säker på, men sen är man ju olika som personer också. Vissa tar mer initiativ än andra helt enkelt

## Frågor

## Svar från testare 3

Hur länge har du varit på företaget?

Snart 1 år.

Hur länge har du jobbat med testning?

Sedan jag började jobba här.

Kan du själv välja hur du skall testa, om ja beskriv när och hur?	Ja till viss mån kan jag det men där finns ju i princip alltid bestämda saker som måste testas. Heuristic testing är dock rätt fritt och man får ta egna initiativ och så.
Vad baserar du dina beslut på när du bestämmer vad som skall testas?	Svår fråga... Jag antar att jag mest baserar dom på... Man märker under själva testningen vad som fungerar dåligt och då under nästa test så tittar man lite extra på det. Man skaffar sig erfarenhet och lär sig vad som oftast brukar krångla och vad man behöver lägga extra tid på.
Hur mycket i procent skulle du säga att du bestämmer och hur mycket är bestämt?	20% kanske? Men det varierar nog... det kan nog vara mer också.
Skulle du kunna bestämma mer själv men väljer du att inte göra det?	Nej det tror jag faktiskt inte. Jag tror det är viktigt att man så mycket som möjligt försöker "tänka" själv, komma med idéer och vara lite självständig också. Annars blir man ju som en robot och gör samma sak hela tiden.
Har det att göra med hur rutinerad man är på hur mycket man som testare beslutar själv?	Självklart är det så. Man blir ju hela tiden bättre på det man gör och man förstår hur saker och ting fungerar. Ju mer rutinerad man är ju snabbare går det också och då kan man kanske "sväva" lite utanför ramen och köra lite testning på andra saker och ting som man kanske märker behöver lite mer fokus.

### Kompletterande fråga:

**Har det blivit någon skillnad i omfattning av mobil kontra stationär testning i företaget när ni jobbade enligt vattenfallsmodellen eller när ni jobbar agilt?**

#### Beslutsfattare 1

Nej det skiljer sig inte något. **Vad tror du det beror på?** Det har att göra med när i processen man befinner sig. I början testas det stationärt men mot slutet testas det nästan enbart mobilt. I början av processen kan man inte testa mobilt då mjukvaran inte finns i mobilen ännu.

#### Beslutsfattare 3

Nej det gör det inte. **Vet du vad det skulle kunna bero på?** Det har att göra med när man kan testa mobilt. Kan man det så behöver man inte testa stationärt.

#### Testare 1

Jag började arbeta på företaget för ungefär två år sedan, vilket betyder att jag kom mitt under förändringen i utvecklingsprocessen och kanske inte kan ge ett bra svar på den frågan. Jag skulle vilja säga att där antagligen inte är några skillnader. **Vad tror du att det kan bero på?** Svår fråga. Tror att dessa aspekter inte är något som påverkas av förändringar i utvecklingsprocessen. Vad som är mobilt och stationärt kan också vara lite svårt att definiera just här, så kan inte ge något heltäckande svar.

## **Testare 2**

Några direkta skillnader tror jag egentligen inte att det är. **Vad tror du att det kan bero på?** Jag tror det är som med all testning här på företaget. Själva utförandet av metoderna, och i det här fallet mobil och stationär testning, tror jag inte förändras. Omfattningen då vill säga. Förändring skulle så fall vara när i processen som tester utförs.

## Referenser

Andrade R, Da Costa A, Dantas V, Marinho, F (2009): Testing requirements for mobile applications. 24th International Symposium on Computer and Information Sciences

Avison D, Fitzgerald G (2006): *Information systems development: methodologies, techniques & tools*. McGraw-Hill Education, Berkshire

Burnstein I (2003): *Practical Software Testing: A process oriented approach*. Springer, cop, New York

Cockburn A (2002): *Agile Software Development*. Pearson Education, Boston

Crispin L (2006): Driving Software Quality: How Test-Driven Development Impacts Software Quality, *Software*. IEEE: 23 Issue: 6 70 - 71

De Lucia A, Ferrucci F, Tortora G, Tucci M (2008): *Emerging Methods, Technologies, and Process Management in Software Engineering*. Wiley, Hoboken

Dubinsky Y, Hazzan O, Keren A, Talby D (2006): Agile Software Testing in a Large-Scale Project, *Software*, IEEE :23 Issue:4: 30 – 37

Eriksson U (2004): *Test av kvalitetssäkring av IT-System*. Studentlitteratur, Lund

Görling S (2009): *Att arbeta med IT-projekt*. Studentlitteratur, Lund

Manifesto for agile software development, <http://www.agilemanifesto.org/> (2009-11-22)

ISO 9126 Software Quality Characteristics, <http://www.sqa.net/iso9126.html> (2009-12-11)

Jacobsen D (2007): *Vad, hur och varför?: Om metodval i företagsekonomi och andra samhällsvetenskapliga ämnen*. Studentlitteratur, Lund

Perry W (2000): *Effective Methods For Software Testing*. Wiley, New York

Petersen K, Wohlin C (2010). *The effect of moving from a plan-driven to an incremental software development approach with agile practices: An industrial case study*. Blekinge Institute of Technology, Blekinge.

Satalkar B (2010): Modified Waterfall Model, <http://www.buzzle.com/articles/modified-waterfall-model.html> (2011-12-01)

Stober T, Hansmann U (2010): *Agile Software Development: Best Practices for Large Software Development Projects*. Springer, New York

Trost J (1997): *Kvalitativa intervjuer, andra upplagan (Qualitative Interviews, second edition)*. Studentlitteratur, Lund