Examensarbete

HÖGSKOLAN I SKÖVDE

PROCESSEN ATT VÄLJA ETT VERKTYG FÖR AUTOMATISERAD GUI-TESTNING

En fallstudie

Examensarbete inom huvudområdet
Datavetenskap
Grundnivå 15 högskolepoäng
Vårtermin 2013

Martin Hallnemo

Handledare: Birgitta Lindström
Examinator: Jonas Mellin

Sammanfattning

Grafiska användargränssnitt (GUI) utgör en viktig del hos många programvaror. En vanlig verifieringsteknik för programvara är testning, vilket ofta upptar mycket tid inom utvecklingsprojekt. En ansats till en effektivare testning med högre kvalitet är testautomation. Att automatisera tester för GUI:n är ofta svårt men värdefullt. Ett vanligt problem är att verktyg med otillräcklig funktionalitet införskaffas för automatiseringen, vilket kan bero på bristfälliga utvärderingar.

I den här rapporten presenteras kriterier som kan användas vid en utvärdering av automatiserade GUI-testningsverktyg, vilka kan öka chanserna för att rätt verktyg införskaffas. Dessa kriterier har identifierats genom en litteraturstudie och fallstudie. Kriterier för urval av verktyg för utvärdering, selektionskriterier, har kategoriserats enligt ISO/IEC 9126, och utvärderingskriterier utifrån testprocessen. Delar av kriterierna har utvärderats i en fallstudie för två Capture & Replay verktyg. Utifrån fallstudien går det att konstatera att åtminstone delar av kriterierna kan användas, och att de ger värdefull information vid en utvärdering.

Nyckelord: programvarutestning, GUI-testning, automatiserad testning, utvärdering av testverktyg, verktygsutvärdering.

Innehållsförteckning

1 Introduktion

Många programvaror har ett grafiskt användargränssnitt (GUI), genom vilket användare kan interagera med programvaran. Det är viktigt att ett GUI har rätt beteende eftersom det styr hur en programvara kan användas. En viktig del inom programvaruutveckling är verifiering som syftar till att kontrollera att en programvara har rätt beteende enligt de krav som finns.

En vanlig verifieringsteknik är testning som innebär att en programvara ges en viss input, och det resultat som fås jämförs med förväntat resultat. Testning är ofta tidskrävande, särskilt för GUI:n. Dels eftersom det kräver att en användare interagerar med olika komponenter i gränssnittet, men även på grund av dess händelsebaserade karaktär. Ett GUI har hela tiden under körning ett tillstånd, vilket består av värdena hos alla GUI-komponenter. Vad som sker för en händelse beror på tillståndet hos GUI:t, vilket i sin tur beror på tidigare exekverade händelser. Ett problem kring den här tillståndsberoende kontexten är att varje händelse bör testas i ett antal olika tillstånd, vilket kan bli tidskrävande (Yuan, Cohen & Memon, 2011). Även för ett relativt litet gränssnitt blir antalet möjliga permutationer av händelser många. Till exempel kan fem olika händelser väljas på 120 sätt om varje händelse måste väljas exakt en gång - det blir snabbt en explosion av antalet inputmöjligheter.

Testautomation kan vara en ansats till en effektivare testning med högre kvalitet. Det finns flera typer av testautomation, med olika syfte och som riktar sig mot olika testaktiviteter. När det gäller GUI-testning går det bland annat att automatiskt generera testfall, samt exekvera tester och analysera de resultat som ges. Testautomation kan innebära en del fördelar, men det finns även risker. Persson och Yilmazturk (2004) redogör för 34 olika risker kring testautomation. En av de risker som nämns är att fel verktyg används för automationen - ett verktyg med otillräcklig funktionalitet. Det finns många verktyg för automatiserad GUI-testning vilka har olika funktionalitet och riktar sig mot olika typer av applikationer. Valet av verktyg kan därför vara svårt. Utvärdering nämns i litteraturen som ett lämpligt tillvägagångssätt för att identifiera rätt verktyg. Det råder dock delade uppfattningar om hur en sådan utvärdering ska gå till och vad som ska utvärderas. Det kan därför vara svårt att veta vad som bör utvärderas.

I det här arbetet har kriterier för utvärdering av automatiserade GUI-testningsverktyg tagits fram, vilket gjorts genom en litteraturstudie och fallstudie. Två typer av kriterier presenteras i den här rapporten: *selektionskriterier*, vilka används för att sålla fram en delmängd av verktyg för utvärdering, samt *utvärderingskriterier* som används för att funktionellt utvärdera verktyg. Selektionskriterierna har delats in enligt ISO/IEC 9126 (kvalitetsstandard för programvara), med utökade kategorier för leverantörskvalifikationer, leverantörssupport, samt licensiering och pris. Utvärderingskriterierna har delats in enligt en abstrakt testprocess. De här kriteriernas relevans har analyserats, och delar av kriterierna har även använts i en utvärdering för två Capture & Replay verktyg, för att visa på att kriterierna kan användas och hur.

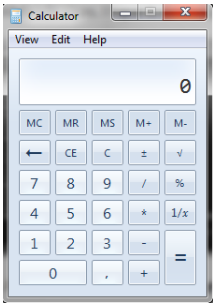
De kriterier som tagits fram kan vara till nytta för den som utvärderar automatiserade GUI-testningsverktyg; de kan förbättra chanserna att rätt verktyg identifieras vid en utvärdering. Utvärderingsmetodiken (intervjufrågor m.m.) som använts kan också vara till nytta och återanvändas i andra utvärderingar.

2 Bakgrund

Grafiska användargränssnitt (GUI) är den vanligaste typen av användargränssnitt för dagens programvara (Brooks & Memon, 2007), då de förser användarna med ett enkelt sätt att använda en programvaras funktionalitet. Memon (2001) ger följande definition på vad ett GUI är:

> *"A Graphical User Interface (GUI) is a hierarchical, graphical front-end to a software system that accepts as input user-generated and system-generated events, from a fixed set of events and produces deterministic graphical output. A GUI contains graphical widgets; each widget has a fixed set of properties. At any time during the execution of the GUI, these properties have discrete values, the set of which constitutes the state of the GUI."* (Memon, 2001, s.25)

Den här definitionen gäller dock inte för alla klasser av GUI-applikationer; för exempelvis icke-deterministiska GUI:n så måste den här definitionen modifieras (Memon, 2001). Vad som dock gäller för alla typer av GUI:n är att de består av ett antal grafiska komponenter (GUI-komponenter), genom vilket användare kan trigga olika händelser som förser GUI-applikationen i ett visst tillstånd. Figur 1 visar ett exempel på en applikation som har ett GUI.



Figur 1. Exempel på applikation med grafiskt användargränssnitt (GUI).

Det är viktigt att ett GUI fungerar som det är tänkt, då det påverkar hur användare kan använda programvaran. En icke-fungerande knapp i ett användargränssnitt kan leda till att personer inte kan utföra sina arbetsuppgifter etc. Ett sätt för att säkerställa kvaliteten hos en programvara är genom att utföra testning som är en typ av verifiering (se 2.1). Ammann och Offutt (2008, s.13) definierar testning som en utvärdering för en programvara, där dess exekvering observeras. Enligt den definitionen framgår det att vad som utvärderas vid testning kan vara olika saker. Funktionell testning är en typ av testning där en programvaras funktionella beteende utvärderas genom att en input ges, som förväntas ge en viss output enligt de krav som finns. Skiljer sig faktisk output från förväntad output så finns det fel hos programvaran, så kallade buggar. Spåra och åtgärda buggar, så kallad debugging, ses vanligtvis inte som en testningsaktivitet. GUI-testning är en typ av testning som syftar till att verifiera att en GUI uppfyller de krav som ställs på det enligt kravspecifikationen. Enligt Brooks och Memon (2007) utgör koden för ett GUI i snitt 40-60% av en programvaras totala kod. Det kan därför finnas många buggar för GUI:t, och inte bara för affärslogiken.

Testning utgör i många fall en tidskrävande och kostsam del inom utvecklingsprojekt. Grindal, Offutt, och Mellin (2006) utförde en studie i vilken det framgick att den genomsnittliga tid som läggs på testning inom utvecklingsprojekt är 35 %, och för säkerhetskritisk programvara kan det handla om så mycket som 65 %. En ansats till att

kunna minska den tid som läggs på testning är genom testautomatisering (se 2.3) (Ramler & Wolfmaier, 2006; Persson & Yilmazturk, 2004). Vissa testaktiviteter är tidskrävande, repetitiva och tråkiga att utföra. Genom att automatisera de testerna kan bland annat mer testning utföras oftare, och dessutom frigörs mänskliga resurser till annat. Att automatisera olika testaktiviteter är dock inte alltid enkelt, särskilt inte om det handlar om GUI-applikationer där en verifiering måste göras både för de visuella elementen som ingår i gränssnittet, samt för de funktionella delarna hos applikationen.

Testning av GUI:n utgör en särskilt tidskrävande och komplex typ av testning. En anledning till det är att GUI-applikationer baseras på händelsebaserad programmering; en GUI-applikation kan ofta uppta många möjliga tillstånd (Yuan, Cohen & Memon, 2011). Ett GUI har hela tiden under exekvering ett tillstånd, vilket innefattar värdena för de grafiska elementen hos GUI:t. Antag att ett GUI består av ett antal checkboxar, som antingen kan vara ifyllda eller tomma. Tillståndet hos hela GUI:t består då av kombinationen av egenskaper för alla checkbox elementen (ifylld/inte ifylld, position etc.). Det tillstånd som en applikation befinner sig i beror på tidigare exekverade händelser, och kan påverka beteendet av en ny händelse. Yuan m.fl. (2011) menar att en svårighet med det är att varje händelse måste testas i flera olika tillstånd, vilket ofta blir tidskrävande. Det blir snabbt en tillståndsexplosion, till exempel kan fem olika händelser väljas på 120 sätt om varje händelse måste väljas exakt en gång. De här 120 sätten att välja händelser på kan leda till 120 olika sluttillstånd. Testningen försvåras även av att det kan vara svårt att observera tillståndet för ett GUI och hur det hamnat i det (Yuan m.fl., 2011).

Testning av GUI:n är även tidskrävande beroende på att det krävs att en användare interagerar med gränssnittet; det finns dock verktyg som möjliggör automatiserad GUI-testning. Exempelvis kan användarinteraktioner spelas in och sedan återspelas vid senare tillfällen, med ett Capture & Replay verktyg. Det finns även modellbaserade verktyg, som möjliggör automatisk generering av testfall (Heiskanen m.fl., 2010).

I resten av det här kapitlet presenteras relevant bakgrundsinformation för det här arbetet, som utförs inom området automatiserad GUI-testning.

2.1 Verifiering och validering

Verifiering och validering är viktiga aktiviteter inom programvaruutveckling, som lätt kan blandas ihop. De här begreppen beskrivs i IEEE-standard 1012 (2012). V*alidering* beskrivs som en process som utförs under eller efter en utvecklingsprocess där en utvärdering görs för om en produkt uppfyller de krav som finns specificerade för den. Validering svarar med andra ord på frågan om rätt produkts har byggts, vilket kräver domänkunskap för att kunna besvara. Antag att en termometerapplikation utvecklats som visar temperaturen i Celsius format, där beställaren har förväntat sig att Fahrenheit ska användas. Genom validering kan domänkunniga (i det här fallet beställaren) konstatera att fel applikation har byggts. Det här kan bero på att fel krav angetts i kravspecifikationen etc.

Verifiering beskrivs enligt samma standard (IEEE 1012:2012) som en process där en utvärdering görs kring om en produkt i ett givet stadium i utvecklingsfasen uppfyller de villkor som ställs på produkten i den fasen. Verifiering svarar alltså på frågan om en produkt byggts på rätt sätt. Kunskapen som krävs för de båda aktiviteterna (verifiering och validering) skiljer sig åt. Verifiering är en mer teknisk aktivitet, som kräver kunskap om den artefakt som verifieringen berör, medan validering kräver domänkunskap (Ammann & Offutt, 2008, s. 11).

En vanlig verifieringsteknik är testning (Grindal m.fl., 2006), som innebär att en programvaras exekvering utvärderas; programvaran förses med en viss input och erhållet resultat (output) jämförs med förväntat resultat. Se exempelvis Amman och Offutt (2008), och Hass (2008). En annan informell verifieringsteknik är inspektion, vilket innebär att en programvaras kod granskas på ett ordnat sätt, se Ackerman, Buchwald och Lewski (1989).

Den finns även formella verifieringstekniker; verifiering som tillskillnad från informell verifiering bevisar något om en programvaras egenskaper. Bevisföring och model checking är exempel på formell verifiering. Bevisföring (engelska: theorem proving) innebär att ett påstående (exempelvis en matematisk formell) bevisas för ett system. Model checking är en annan typ av formell verifiering där olika egenskaper kontrolleras på en representativ modell (ändlig tillståndsmaskin etc.) av ett system, se exempelvis Berezin (2002)

Det finns många verifieringstekniker (fler än ovannämnda), med olika för- och nackdelar. Exempelvis kan bevisföring bevisa något om en programvaras egenskaper, vilket ej testning kan (det går inte att testa allting). En nackdel med bevisföring är att det kan vara svårt, och om det blir fel kan det leda till en falsk trygghet. En fördel med testning är att den faktiska programvaran körs, vilket inte görs för alla typer av verifiering. Val av verifieringsteknik bör göras efter situationen; beroende på programvara, tid, och kunskap etc. Gemensamt för all verifiering är dock dess syfte - att bidra till programvara av högre kvalitet.

2.2 Testning

Testning är som nämnts i 2.1 en vanlig typ av verifiering för programvara, där dess exekvering utvärderas. Vad syftet med testning är råder det dock delade uppfattningar om. Ammann och Offutt (2008, s.8) redogör för olika synsätt på testning inom organisationer. Det mest optimala synsättet på testning enligt dem är att testning är en mental disciplin som bidrar till att programvara med högre kvalitet utvecklas. Den sämsta nivån innebär att ingen skillnad görs på testning och debugging.

Förutom olika sätt att se på testning så finns det även olika typer av testning, som syftar till att verifiera olika saker hos en programvara. Vilken typ av testning som utförs och hur valet av vad som ska testas går till är faktorer som påverkar kvaliteten hos testningen. Testning kan utföras med olika grad av struktur, alltifrån att tester utförs slumpmässigt, till att de utförs efter noggrant uppsatta riktlinjer och testkrav. Grindal m.fl. (2006) har gjort en studie som visar att formella strategier för hur testningen ska gå till sällan används inom organisationer. Ett av de motiv som lyfts fram till varför det är så, är att så länge fördelarna med en förbättrad testning är oklara så finns det lite motivering till förändring. En fördel med att använda strukturerad testning är att det går det att uttala sig om hur en programvara testats, t.ex. att all kod körts åtminstone en gång. Det är dock viktigt att vara medveten om att testning aldrig bevisar att ett program har vissa egenskaper (exempelvis att den är felfri). Det beror på att det inte går att testa alla möjligheter för en programvara.

I följande sektioner beskrivs testkrav, testfall, testorakel och testprocessen som utgör viktiga delar hos testningen.

2.2.1 Testkrav

Testkrav är konkreta, tydligt specificerade krav som ska uppnås genom testning (Ammann & Offutt, 2008, s.17). Testkrav kan rikta sig mot olika programvaruartefakter, till exempel en programvaras källkod eller inputrymd. Exempel på testkrav är att en viss rad i källkoden för

en programvara ska täckas av minst ett test, eller att ett logiskt uttryck på en viss rad ska evalueras till sant av minst ett test. Testkrav är alltså en abstrakt beskrivning som anger vad man vill uppnå med testningen, utan att tala om hur.

Ett sätt för att ta fram testkrav för en programvara är genom att använda formella täckningskriterier. Ett täckningskriterium definieras av Ammann och Offutt (2008, s.17) som en eller ett antal regler som måste uppfyllas för ett testset. Täckningskriterier anger alltså hur testkrav ska tas fram. Kodtäckning är en typ av täckningskriterier som ställer krav på hur källkoden för en programvara ska testas (att all kod ska köras etc.). Predikattäckning är ett konkret exempel på täckningskriterium, vilket har testkraven att varje logiskt uttryck i en kod ska evalueras till sant, respektive falskt för minst ett test. När det gäller testning av GUI:n är dock inte kodstrukturella testkrav särskilt passande (Memon m.fl., 2001). Det beror på att vad som sker vid en viss händelse beror på tillståndet hos GUI-applikationen (som i sin tur beror på tidigare utförda händelser), vilket medför att varje händelse bör testas inom ett antal olika tillstånd (Yuan m.fl., 2011). Samverkan mellan händelser är alltså centralt vid testning av GUI-applikationer.

Yuan m.fl. (2011) beskriver att en vanlig teknik för att ta fram testkrav för ett GUI är att modellera det som en riktad graf, där varje nod representerar en händelse, och varje väg en tillståndsövergång (en så kallad EventInteractionGraph). Testkrav kan sedan specificeras utifrån modellen. Exempelvis att alla händelsesekvenser av en viss längd ska testas. Yuan och Memon (2010) har studerat hur antalet ingående händelser för ett test påverkar dess förmåga att upptäcka fel. De konstaterar att korta sekvenser (två eller tre inkluderade händelser) är mest effektiva för att upptäcka fel. Ett problem med längre sekvenser är att antalet möjligheter blir många, vilket gör att det blir opraktiskt att testa alla kombinationer.

2.2.2 Testfall

Testfall är vad som behövs för att kunna uppfylla ett testkrav. Ett testfall består av input till programvaran som testas (konkreta parametervärden etc.), förväntat resultat (enligt kravspecifikation), samt nödvändiga pre- och postfixvärden för att testet ska kunna utföras och utvärderas (tillståndet hos en GUI-applikation etc.) (Ammann & Offutt, 2008, s.15). Testning innebär alltså att testfall körs på en programvara för att det ska gå att utvärdera om den har rätt beteende. Ett testfall kan uppfylla ett eller fler testkrav, men för att uppfylla den totala mängden testkrav för en programvara så behövs ofta flera testfall. En sammanhörande mängd testfall kallas för en testsvit. Predikattäckning är som nämnt ett täckningskriterium, med testkraven att alla logiska uttryck för en programvara ska evalueras till sant, respektive falskt för minst ett testfall. I Tabell 1 ges ett exempel på hur Predikattäckning kan uppfyllas för exempelkoden i Figur 2.

```
if(x > 0 or y > 0){
      //Gör någonting.
}
```

Figur 2. Exempel på logiskt uttryck i pseudokod.

Tabell 1. Exempel på hur testkriteriet Predikattäckning kan uppfyllas för Figur 2.

P1 (x > 0)	P2 (y > 0)	P (x > 0 or y > 0)	Input (x, y)
T	T	T	(2, 7)
F	F	F	(-1, -3)

- Varje rad i tabellen representerar ett testkrav

När det gäller testning av GUI:n så menar Xie och Memon (2006) att den vanligaste strategin är att testa efter vad som ”känns rätt” och att testningen avslutas när det inte finns mer tid, så kallad ad hoc testning. Det beror framförallt på den komplexitet som testning av GUI:n innebär, samt att det inte finns några vedertagna täckningskriterier som lämpar sig för GUI-testning. Att designa testfall för GUI:n är ofta ett svårt och tidskrävande jobb; val av testsekvenslängd och vilka händelser som ska ingå i testfall är sällan uppenbart. Autogenerering av tester genom ett testverktyg kan därför vara värdefullt (Memon, Soffa & Pollack, 2001). De verktygen baseras på modellbaserad testning och beskrivs mer i 2.4.2.

2.2.3 Testorakel

När en applikation testas är det betydelsefullt vilken output som ges. Ett sätt att verifiera outputen på är genom att använda testorakel. Ett testorakel består av två delar: orakelinformationen som är det förväntade resultatet, samt orakelproceduren som jämför förväntat resultat med faktiskt resultat (Xie & Memon, 2007).

Ett testorakel kan utformas på flera sätt, genom att olika orakelinformation och orakelprocedurer används. Xie och Memon (2007) exemplifierar det här genom att nämna att ett Excel-ark kan verifieras på flera sätt; värdet för en enskild cell kan avläsas och jämföras med det förväntade värdet, eller den sammanlagda summan för alla celler kan jämföras med den totala förväntade summan. Det finns olika för- och nackdelar med att använda ett visst orakel; kostnaden för att beräkna och underhålla orakelinformation varierar, vilket även är fallet för kostnaden att implementera och exekvera orakelproceduren. När det gäller testorakel för GUI:n så måste det påkallas på fler ställen än för sluttillståndet för ett testfall, då en felaktig skärm kan leda till ett felaktigt tillstånd (Memon, 2001).

Ett testorakel kan antingen vara manuellt eller automatiserat, principen är dock den samma; erhållet resultat jämförs med förväntat resultat. Xie och Memon (2007) nämner att i praktiken används fyra tillvägagångssätt för att skapa testorakel för GUI-applikationer:

- Manuellt orakel – Den mest populära typen av orakel som innebär att en testare manuellt interagerar med ett GUI och därefter visuellt observera att det inte finns fel.

- Capture & Replay – Innebär att ett verktyg används för att spela in en användarinteraktion, vilken sedan kan återspelas med verktyget. Testaren ser till att ett visst tillstånd eller respons från GUI:t ingår i det inspelade scriptet, vilket fungerar som förväntat resultat vid återuppspelning.

- Programmerade tillståndskontroller - Innebär att explicita kontroller av olika tillstånd görs under exekvering (engelska: asserts). Uppfylls inte det villkor som kontrolleras avbryts exekveringen. Figur 3 visar en tillståndskontroll för variablerna x och y i C:

```
#include <assert.h>
            assert(x > y);
```

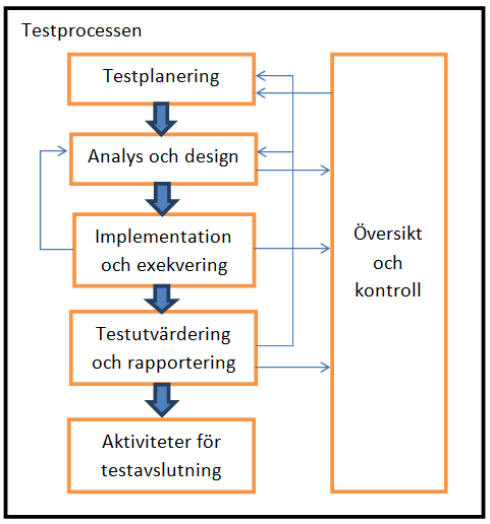
Figur 3. Kontroll av tillståndet hos variablerna x och y vid exekvering (att x är större än y).

- Test harness – Är en testmiljö bestående av bland annat stubs och drivers som behövs för att ett test ska kunna köras. En test harness möjliggör att en användarinteraktion kan simuleras, genom att metodanrop görs till olika händelser, utan att en användare interagerar med GUI:t. För att skapa testorakel kan förväntat resultat hårkodas i de script som körs av en test harness.

En blandning av tekniker kan även användas, exempelvis kan en användarsession spelas in, för att sedan editeras manuellt för att förbättra dess kvalitet. Gemensamt för alla tekniker för att skapa orakel är att de kräver en betydande del manuellt arbete (Xie & Memon, 2007).

2.2.4 Testprocess

Testningsarbetet inom organisationer utförs ofta enligt en testprocess som anger hur testningen ska utföras. Hur en sådan testprocess ser ut kan skilja sig åt mellan organisationer. Det finns abstrakta testprocesser som beskriver hur en testprocess kan se ut.



Figur 4. Den allmänna testprocesssen enligt ISTQB* , baserad på Hass (2008, s.37).

Den allmänna testprocessen enligt ISTQB (International Software Testing Qualifications Board) illustreras i Figur 4. Den kan beskrivas som en process som syftar till att ge information för att säkerställa kvaliteten av en produkt, beslut, och processerna inom ett testningsarbete (Hass, 2008, s.35). Här följer en beskrivning av de olika stegen.

I *testplanering* (Hass, 2008, s.39) ska planering av testningsarbetet göras. Det här omfattar bland annat att definiera vad som ska testas, hur resurser ska användas kring testningen, vilka tillvägagångssätt som ska användas för olika testaktiviteter (för att ta fram testfall etc.) med mera. Testplaneringen ska resultera i att övergripande dokumentation tas fram för testningsarbetet. A*nalys och design* (Hass, 2008, s.50) syftar till att utifrån dokumentation från testplaneringen designa testfall och testmiljö. När det gäller testfall så ska först högnivå testfall designas, vilket är testfall som beskriver vad som ska testas i generella drag utan angivna värden för input och förväntat resultat. De här ska sedan konkretiseras i lågnivå testfall, där både input och förväntat resultat ska finnas angivet. Testfall utvinns vanligtvis från kravspecifikationer, därför är det viktigt att i det här skedet skapa en spårbarhet över hur testfallen mappar mot krav. När det gäller analys så är det viktigt att analysera hur programvaran testas utifrån avsedda testfall, samt huruvida testfallen kan förbättras.

Implementation och exekvering (Hass, 2008, s.61) syftar till att lågnivå testfall ska konkretiseras i form av testbeskrivningar (manuell testning) eller testscript (automatiserad testning). Det här ska möjliggöra att tester kan exekveras. I det här skedet ska även testmiljön göras i ordning, vilket bland annat omfattar den fysiska miljön som testerna ska utföras i, hårdvara, verktyg m.m. Viktiga faktorer för testexekvering är att erhållet resultat

* www.istqb.org/downloads/viewcategory/16.html - se Foundation Level Syllabus.

jämförs med förväntat resultat, att tester kan återupprepas för regressionstestning eller i bekräftelse syfte, att testresultat loggas m.m. *Testutvärdering och rapportering* (Hass, 2008, s.72) syftar till att göra övergripande utvärdering av den testning som utförts . I det här skedet ska bland annat en utvärdering göras över om testning ska avslutas eller ej, baserat på det kriterium som satts upp för när testning kan anses klar. Testrapporter ska även skapas, för att förse en spårbarhet över vilka tester som utförts och dess utfall.

Slutligen ska *aktiviteter för testavslutning* utföras (Hass, 2008, s.74). I det här skedet ska erfarenheter över testningen dokumenteras för att underlätta för framtida testning. Testartefakter ska även arkiveras och eventuellt överlämnas till någon. Innan testningen avslutas är det viktigt att kontrollera ytterligare en gång att de krav som specificerats för när testningen kan avslutas har uppfyllts. Dessutom ska dokumentation göras över hur många tester som körts och hur många som lyckats, hur lång tid testningen har tagit m.m. De här värdena ska även jämföras med det förväntade.

Testprocessen i Figur 4 är iterativ. Det här innebär att vissa steg kan behöva genomgås flera gånger. Exempelvis om det visar sig vid implementation att avsedda tester inte går att automatisera, så måste en ny analys göras, och nya testfall designas. När det gäller testplanering och kontroll (Hass, 2008, s.40) så är det en aktivitet som ska utföras parallellt med övriga steg i testprocessen, där framstegen i testningsarbetet ska jämföras med den övergripande planen som finns för testningen. Nödvändiga ändringar i testningsarbetet ska även göras, exempelvis om planen som tagits fram inte kan följas av olika anledningar.

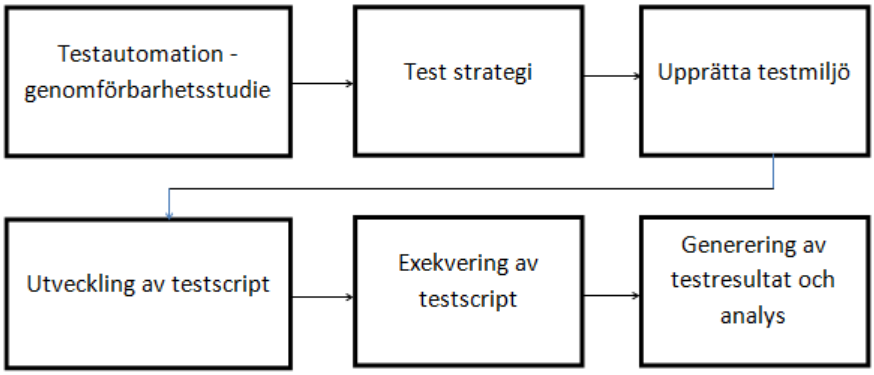
2.3 Automatiserad Testning

Det går att automatisera flera olika typer av testning. Wissink och Amaro (2006) anger följande kategorier av testautomation: testskötsel, enhetstestning, testfallsgenerering, prestandatestning, och funktionell testning. Testautomation kan till exempel möjliggöra att en mängd tester kan köras utan att någon måste manuellt utföra och utvärdera resultat av varje test. Genom automatisering kan bland annat testingskostnaderna inom ett projekt sänkas, mer testning kan utföras under en viss tidsperiod m.m. (Hicks, South & Oshisanwo, 1997; Ramler & Wolfmaier, 2006). Regressionstestning är särskilt lämplig att automatisera, då det är repetitiva och tidsödande tester (Memon, 2008).

I Figur 5 illustreras hur automatiseringsprocessen kan se ut (Namachivayam, 2012). Processen att automatisera bör inledas med en analys kring om det går att automatisera testning för den eller de applikationer som avses. Den här analysen bör bland annat inbegripa om avsedda testfall går att automatisera, samt vilka automatiseringsverktyg som kan vara aktuella. När en genomförbarhetsanalys har gjorts så ska en teststrategi tas fram, vilken anger hur automatiseringen ska gå till. Det här omfattar bland annat val av ramverk för testautomatisering, tidsplanering, fördelning av resurser inom projektet m.m. En analys bör även göras över automatiseringens återbetalning. När en teststrategi tagits fram ska testmiljön upprättas (testverktyg ska införskaffas och installeras etc.)

När testmiljön har upprättats ska automatiserade tester, i form av testscript, tas fram (Namachivayam, 2012). Det kan göras på olika sätt (se 2.4). Det mest fundamentala i automatiseringsprocessen är att automatisera testexekveringen, eftersom den största återbetalningen kan fås där (Wissink & Amaro, 2006). Viktiga faktorer kring den automatiserade testexekveringen är att en automatisk loggning görs för exekverade tester och

dess utfall för att ge en spårbarhet, samt att förväntat och faktiskt resultat jämförs. Slutligen är det viktigt att automatisera analysen av tester (huruvida testerna lyckats etc.) och att testrapporter genereras.



Figur 5. Processen för testautomatisering, baserad på Namachivayam (2012).

Att automatisera testning är ofta svårt, och inte alltid lönsamt. Lära sig att automatisera, utföra den, samt utvärdera resultatet av den automatiserade testningen tar tid och kostar pengar (Ramler & Wolfmaier, 2006). Persson & Yilmazturk (2004) redogör för 34 olika risker med testautomation, bland annat nämns okontrollerad införsel av testautomatisering i en omogen organisation som en risk; ”*automated chaos yields faster chaos*”. Andra risker som nämns är bland annat att mycket tid måste läggas på underhållsarbete, att testverktygen inte har tillräckligt funktionalitet, att fel tester automatiseras m.m. Gällande GUI-applikationer kan det vara särskilt svårt att skapa hållbara automatiserade tester. Automatiserade tester för GUI-applikationer kan vara känsliga för ändringar i gränssnittet, och det kan även vara svårt att avgöra vilka tester som ska automatiseras (se 2.2.2).

2.4 Tillvägagångssätt testning av GUI-applikationer

Mycket av dagens testning av GUI:n utförs ad hoc (Memon, 2002). Det finns flera tekniker för att testa GUI-applikationer, men faktum är att det inte finns någon fulländad teknik, vilket framförallt beror på den komplexitet som testning av GUI:n innebär (Yuan m.fl., 2011).

Ett tillvägagångssätt för GUI-testning är testning med verktygsstöd. Testning av GUI:n är som nämnts tidskrävande. Ett antal testverktyg har utvecklats bland annat för att kunna minska den tid som måste läggas på GUI-testning. Vad de här verktygen kan utföra varierar. En del kan endast utföra automatiserad testexekvering, medan andra även kan utföra generering av testfall, och konstruktion av en modell över ett GUI genom så kallad reverse engineering (Memon, Banerjee & Nagarajan, 2003). Automatiserade testverktyg bör enligt Lewis (2009, s.399) förankras i en testprocess, som i sin tur ska ingå i en utvecklingsprocess, för att användandet av testverktyg ska vara till nytta. Att använda testverktyg innebär inte succé per automatik; det spelar minst lika stor roll hur verktygen används. Valet av testverktyg dock är kritiskt för en lyckad automation (Persson & Yilmazturk, 2004).

Här följer en beskrivning av två vanliga typer av testverktyg för GUI-testning, samt för vad ett ramverk för testautomatisering är.

2.4.1 Capture & Replay

Capture & Replay verktyg (svenska: spela in och spela upp) möjliggör att det en användare gör med mus och tangentbord i ett GUI spelas in till ett testscript (Börjesson & Feldt, 2012).

Det går sedan att spela upp samma interaktion med verktyget, där samma resultat ska fås som vid inspelningen. Tekniken möjliggör att ett GUI kan bearbetas utan att det behövs en människa som gör det, och det används framförallt för regressionstestning.

Användarsessioner lagras i testscript i form av koordinater för händelserna, eller genom referenser (t.ex. till ett XML-objekt) till berörda GUI-komponenter (Börjesson & Feldt, 2012). Referenser kan vara bättre, eftersom det gör testscripten mindre känsliga mot ändringar hos GUI:t. Förändras layouten hos GUI:t kan GUI-komponenter ändå identifieras i vissa fall. Enligt Memon och Soffa (2003) tar det ungefär 20-30 minuter att skapa ett testfall där 50 olika händelser ingår. Det är därför önskvärt att automatiserade testfall kan återanvändas för framtida versioner av en programvara. Testscript är dock känsliga för ändringar i GUI:t, även om referenser används. Ändras typen på en GUI-komponent från t.ex. en checkbox till en listbox kan det innebära att objektet inte kan identifieras (Grechanik, Xie & Fu, 2009). Att Capture & Replay verktyg ofta kräver mycket manuellt arbete vid skapande och underhåll av testscript, nämns som en av nackdelerna med att använda de verktygen för testautomatisering (Persson & Yilmazturk, 2008).

2.4.2 Modellbaserade testverktyg

Modellbaserade testverktyg är verktyg som möjliggör automatiserad testfallsgenerering (Heiskanen m.fl., 2010; Arlt, Bertolini & Schäf, 2011). Den modellbaserade testprocessen börjar med att en modell skapas, som representerar den programvara som avses ur ett visst perspektiv (ändlig tillståndsmaskin etc.). Utifrån modellen kan sedan testfall genereras med verktyget utifrån ett visst täckningskriterium (se 2.2.1).

Det finns flera fördelar med modellbaserad testning jämfört med scriptbaserad (Heiskanen m.fl., 2010). Dels, så länge modellen överensstämmer med den applikation som ska testas så kan komplexa och användbara testfall genereras automatiskt – testfall som skulle vara svåra för en människa att designa. En annan fördel är att själva modelleringsprocessen kan innebära att många fel upptäcks. En nackdel med modellbaserad testning är att det skalar dåligt för stora system; antal genererade tester kan bli väldigt många, även för simpla täckningskriterier. En strategi för att hantera det är att använda slumpgenerering av tester. Det finns studier som tyder på att skillnaden mellan att använda slumptestning och strukturerad testning kan vara liten, vilket Claesson och Hughes (2011) diskuterar.

2.4.3 Ramverk för testautomatisering

Testning av GUI:n består enligt Memon (2001) av följande steg som måste utföras: (1) att avgöra vad som ska testas, (2) att ta fram testfall, (3) att ta fram förväntat resultat för testfall, (4) att exekvera testfall och verifiera output, (5) att utföra regressionstestning, (6) att avgöra om ett GUI har testats ordentligt. Många av de testverktyg som finns tillgängliga kan endast utföra en eller ett par av de här uppgifterna. Följaktligen kan det finnas intresse av att utveckla eller använda verktyg som kombinerar ovan nämnda steg. Memon (2001) beskriver att ett ramverk för testautomatisering är en kombination av de verktyg, tekniker och riktlinjer som behövs för att kunna utföra alla steg vid automatiserad testning av ett GUI.

Det finns många fördelar med att använda ett ramverk för testautomation, till exempel att det spar resurser (mindre manuell testning måste genomföras) och underlättar i automatiseringsarbetet. Det finns dock en del svårigheter med att skapa ett sådant ramverk. Exempelvis måste en allmän GUI-representation skapas som täcker merparten av GUI:n, för att det ska kunna appliceras på flera typer av GUI:n (Memon, 2001). En annan utmaning är hur testfall ska genereras; alla kombinationer är sällan lämpligt.

2.5 Att välja testverktyg

Valet av verktyg för testautomatisering är kritiskt, men inte alltid enkelt (Persson & Yilmazturk, 2004; Hass, 2008, s.362). Verktygen har olika funktionalitet, och ställer olika krav på användaren. En utvärdering kan därför vara bra att utföra, för att identifiera ett så passande verktyg för situationen som möjligt. Det finns flera sätt att utföra en utvärdering på, och här beskrivs några.

Poston och Sexton (1992) anser att selektionskriterier (kriterier för gallring av kandidatverktyg) ska kunna sättas upp genom enkätundersökningar. De här selektionskriterierna ska matcha organisationens behov. Enkäterna ska även resultera i att ett antal utvärderingskriterier kan sättas upp. Kriterier för utvärdering delas enligt Poston och Sexton (1992) lämpligtvis in på följande sätt: 1) generella kriterier (vad man vill få ut av verktyget), 2) miljöberoende kriterier (tillgänglig budget för verktyg etc.), 3) funktionella kriterier för verktyget, 4) icke-funktionella kriterier för verktyget (användarvänlighet etc.). Kriterierna kan bedömas procentuellt för varje verktyg, i form av grad av uppfyllelse (0-100%). När alla kandidater utvärderats väljs ett verktyg ut för att användas i ett pilotprojekt.

Hass (2008, s.361) menar att det första som bör göras vid en utvärdering är att reda ut vilka behov som finns kring testautomation och ett sådant verktyg. Finns det ett behov ska ett utvärderingsteam inrättas. Utvärderingsteamets första uppgift är att ta fram en strategi för hur verktyget ska användas inom organisationen (på projektnivå eller processnivå etc.) Sedan ska en bidragsanalys göras för verktyget, där bidraget jämförs med kostnaderna. Krav på verktyget ska även identifierats, både funktionella och icke-funktionella. Utifrån dessa krav ska en lista med kandidatverktyg kunna sammanställas. När alla kandidatverktygen utvärderats, ska de två som presterat bäst utvärderas ytterligare i praktiken. Hass (2008, s.366) ger förslag på att kriterierna kan bedömas enligt grad av uppfyllelse (helt, nästan, delvis, inte), eller procentuellt (0-100%).

Börjesson och Feldt (2012) har gjort en utvärdering för två GUI-testningsverktyg där de matchar sina utvärderingskriterier mot kvalitetskriterier som tas upp i ISO/IEC 9126. Den här matchningen förespråkas även av Illes m.fl. (2005), som har utökat ISO/IEC 9126 med ytterligare kriterier för leverantörskvalifikationer, support, samt licensiering och pris. Illes m.fl (2005) definierar även funktionella utvärderingskriterier för testverktyg genom att utgå från en testprocess. Kategorierna av kriterier ser ut enligt följande: 1) testplanering och kontroll, 2) design av testfall, 3) konstruktion av testfall, 4) testexekvering, 5) fånga och jämföra resultat, 6) testrapportering, 7) spårbarhet testinformation, 8) skötsel av testmiljön.

Behkamal m.fl. (2009) har också använt sig av ISO/IEC 9126 i sitt arbete som handlar om utvärdering av B2B-applikationer. De har inlett sitt arbete med att diskutera olika sätt att kategorisera egenskaper hos programvara på, däribland kvalitetsmodeller och standarder som McCalls, Boehms, ISO m.fl. De har valt att använda ISO/IEC 9126 i sin utvärdering, vilket de motiverar med att det är en universell kvalitetsstandard som lämpar sig att utvinna utvärderingskriterier från och som är relevant för all typ av programvara. Dessutom representerar det kvalitet ur en användares perspektiv.

Sammanfattningsvis är det viktigt att vara medveten om vilka aspekter hos en programvara som täckts in av en utvärdering. Det är bra att dela in kriterier i kategorier för att tydliggöra vad de syftar till att utvärdera. Kriterierna kan utvärderas på flera sätt, i vissa fall kan de kvantifieras, medan andra endast går att bedöma.

3 Problemdefinition

Syftet med det här arbetet är att ta fram ett flexibelt utvärderingsverktyg, innehållande kriterier, som är relevanta och kan användas vid en utvärdering av verktyg för automatiserad GUI-testning. Det här verktyget ska öka chanserna till en lyckosam utvärdering.

Testning är en viktig verifieringsaktivitet inom programvaruutveckling (se 2.1 och 2.2), men som i många fall är tidskrävande och utgör en stor kostnadsfaktor (Grindal m.fl, 2006; Ramler & Wolfmaier, 2006). Testautomatisering med hjälp av verktyg kan därför vara eftertraktat, då det bland annat kan sänka testningskostnaderna inom projekt och öka kvaliteten på testningen (Hicks m.fl, 1997; Ramler & Wolfmaier, 2006). Testning av GUI:n är särskilt viktigt (se 2.4), då de styr hur en programvara kan användas. Att automatisera testningen av GUI-applikationer är dock svårt i många fall. Det beror bland annat på att det ofta finns många möjliga tillstånd, och att det kan vara svårt att skapa ett automatiserat orakel (se 2.2.3). Den här komplexiteten gör att valet av verktyg blir än mer viktigt. En utvärdering är därför lämplig.

Ett problem som har konstaterats kring testautomation i allmänhet är att det verktyg som används för automatisering har otillräcklig funktionalitet, beroende på avsaknad av utvärdering eller att den som utförts varit bristfällig (Person & Yilmazturk, 2004). Problemet med det som tas upp i litteraturen är att det inte är uppenbart hur de olika kriterierna kan användas, samt vilka som är relevanta, då det skiljer sig åt mellan litteratur.

3.1 Problemprecisering

Problemet kan preciseras med följande frågeställning:

- Vilka kriterier är relevanta för en utvärdering av verktyg för automatiserad GUI-testning och hur kan dessa kriterier användas?

Frågeställningen motiveras av att det finns en oklarhet i vilka kriterier som är relevanta för en utvärdering av verktyg för automatiserad GUI-testning. Att automatisera testning kan ge flera fördelar exempelvis sänkta testningskostnader (se 2.3), särskilt för testning av GUI:n som är en viktig, men tidskrävande och komplex typ av testning. Det finns även risker med testautomation, t.ex. att fel verktyg används för automationen vilket gör den dyr eller försämrar dess kvalitet. Valet av verktyg är därför viktigt (se 2.5), då deras funktionalitet varierar. Verktygen kan även rikta sig mot olika testaktiviteter, och typer av GUI-applikationer etc. En utvärdering där olika kriterier utvärderas kan därför vara värdefullt. Vid en sådan utvärdering behövs kriterier för val av verktyg att utvärdera (selektionskriterier), samt utvärderingskriterier som anger vad som ska utvärderas för dessa.

Frågeställningen ovan anses besvarad när följande delmål uppfyllts:

- Delmål 1 – Att identifiera och presentera selektions- och utvärderingskriterier för utvärdering av GUI-testningsverktyg.
- Delmål 2 – Att analysera relevansen hos de kriterier som identifierats.
- Delmål 3 – Att utvärdera de relevanta kriterierna genom att visa på hur de kan användas i praktiken.

4 Metodbeskrivning och tillvägagångssätt

I det här kapitlet diskuteras alternativa och valda metoder för respektive delmål. Dessutom ges en beskrivning av tillvägagångssätt.

- Delmål 1 – Att identifiera och presentera selektions- och utvärderingskriterier för utvärdering av GUI-testningsverktyg.

Kriterier som kan användas vid en utvärdering kan identifieras på flera sätt. En litteraturstudie kan utföras, där en sammanställning görs över vad andra anser är relevanta kriterier vid en utvärdering. Fördelar med en litteraturstudie är framförallt att det belyser vad andra gjort inom området, och på så sätt tydliggör vad som är känt sedan tidigare och vad som är nytt. Nackdelar med litteraturstudie är att det som tas upp i litteraturen inte behöver vara relevant eller pålitligt. Det är därför viktigt att kritiskt granska den.

Andra alternativ för att samla in kriterier är intervjuer eller enkätundersökning. Det här kan göras både med en kvantitativ eller kvalitativ ansats. En kvantitativ ansats innebär att stängda frågor ställs, där svarsalternativen är förutbestämda (exempelvis ja/nej). En fördel med kvantitativa frågor är att svaren blir enklare att analysera, genom att de kan kvantifieras och eftersom statistik kan tillämpas på dem. En nackdel med kvantitativa frågor är att de inte fungerar så bra i utforskande syfte, där inget givet svar finns. En kvalitativ ansats innebär frågor av utforskande karaktär, där svaret inte är givet (ex. vad anser du om x?). Fördelen med kvalitativa frågor är att svar kan motiveras, och att de kan ställas när vad som efterfrågas inte är givet. En nackdel med kvalitativa frågor är att det kan vara svårt att kvantifiera och tillämpa statistik på svaren på ett tillförlitligt och meningsfullt sätt.

Enkäter och intervjuer har olika för- och nackdelar. Enkäter möjliggör att data, i det här fallet kriterier för utvärdering, kan samlas in från många personer under kort tid. Dessutom kan en större sekretess från kring svaren gentemot intervjuer. En nackdel med enkäter är att de kan upplevas jobbiga att fylla i (mycket att skriva etc.). Vid intervjuer kan personer prata om något, utan att behöva tänka på att få det nedskrivet. En annan fördel med intervjuer är att det är enklare att anpassa sig till de svar som ges, exempelvis ställa följdfrågor.

Observationer är ett annat sätt att samla in kriterier på. Det här kan exempelvis ske genom att använda ett par GUI-testningsverktyg och observera saker hos dessa. En fördel med observationer är att genom observera något med egna ögon så kan en ökad förståelse fås för det som avses. En nackdel är att det kan vara svårt att veta vad man ska kolla efter.

För att utvinna kriterier valdes litteraturstudie, intervjuer, och observationer som metoder. Flera metoder valdes av den anledningen att vad teorin och vad som förespråkas i praktiken kan skilja sig åt; flera perspektiv kan därför ge mer pålitliga resultat. Litteraturstudien ger ett teoretiskt perspektiv och knyter an till andras arbeten, medan intervjuer och observationer ger ett praktiskt perspektiv. Intervjuerna och observationerna utfördes i en fallstudie, vars tillvägagångssätt beskrivs i 4.2. Litteraturstudien tillvägagångssätt beskrivs i 4.1.

- Delmål 2 – Att analysera relevansen hos de kriterier som identifierats.

Relevansen hos de kriterier som identifierats kan analyseras på olika sätt. Ett sätt är att göra en enkätundersökning där personer får värdera på en skala för varje kriterium, hur relevanta

de anser att kriterierna är. Är de personer som tillfrågas kunniga kring GUI-testningsverktyg kan det ge någorlunda pålitliga svar. Pålitlighet är annars ett problem för de metoderna.

Litteraturstudie är ett annat alternativ för att analysera relevansen hos kriterierna. En sådan analys kan exempelvis omfatta vad som sägs i litteraturen om kriterierna, eller varför de kan anses relevanta baserat på litteraturen. En fördel med litteraturstudie är som sagt att man knyter an till andras arbete och ger dem ett erkännande för vad de gjort. En nackdel i det här fallet är att det eventuellt kan vara svårt att relatera vissa kriterier till litteraturen.

En analys utifrån litteraturen valdes som metod för det här delmålet (se 4.1). Litteraturstudie användes som en av metoderna för att utvinna kriterier i delmål 1, därför är det lämpligt att återknyta till litteraturen här. I den litteratur som kriterier utvunnits från, finns eventuellt diskussioner kring varför kriterierna är relevanta.

- Delmål 3 – Att utvärdera de relevanta kriterierna genom att visa på hur de kan användas i praktiken.

Ett sätt för att undersöka hur kriterierna kan användas, är att göra en litteraturstudie för att se om andra har använt kriterierna, och i så fall hur. Det kan vara fördelaktigt att ta del av andras erfarenheter av olika kriterier En nackdel med litteraturstudie är att det kan vara svårt att veta varför kriterierna använts på ett visst sätt etc.

En fallstudie kan användas även för det här delmålet, exempelvis genom att använda kriterierna vid en utvärdering. Att befinna sig i problemsituationen kan leda till en ökad förståelse för problemet. I det här fallet kring hur kriterier för utvärdering kan användas. En nackdel med fallstudie är att resultaten kan bli väldigt knutna till ett specifikt fall. Exempelvis hur kriterierna används kan bli väldigt präglat av de verktyg som använts i fallstudien. Andra alternativ för att undersöka hur kriterierna kan användas omfattar bland annat att använda intervjuer eller enkäter för att fråga andra kring hur de tycker att kriterierna kan användas. Problemet med det kan vara att bedöma pålitligheten hos svaren.

En fallstudie valdes som metod för att utvärdera kriterierna. Fallstudie är lämpligt då det återspeglar hur det kan se ut i praktiken. Fallstudien tillvägagångssätt beskrivs i 4.2.

4.1 Litteraturstudiens tillvägagångssätt

Litteraturstudien utfördes på ett strukturerat sätt för att underlätta i sökprocessen. Ett antal relevanta databaser inom ämnesområdet datavetenskap valdes ut, i vilka litteratur söktes fram med hjälp av nyckelord. De databaser som valdes ut var följande: ACM Portal, IEEE Xplore, ScienceDirect, Scopus, samt WorldCat Local. Den främsta orsaken till att de här databaserna valdes ut är att de omfattar det mest av det som publicerats inom datavetenskap. Det finns även andra anledningar till varför just dessa valdes. Exempelvis har IEEE en thesaurus vid sökning som gör det enklare att välja sökord, vilket ansågs vara en fördel. En fördel med Scopus är att det finns ett referenshanteringssystem som är tydligt och enkelt att använda. När det gäller WorlCat Local så användes den framförallt för att hitta relevanta böcker inom området, vilket har fungerat som komplement till forskningsartiklar.

Vid litteratursökningen gjordes en gallring av sökresultaten. Först valdes relevant litteratur ut baserat på titel. Därefter gjordes en mer noggrann granskning av utvald litteratur. Först lästes abstrakt och om materialet ansågs relevant så lästes även innehållet. Dessutom

kontrollerades referenslistorna bland den relevanta litteraturen, med syftet att identifiera ytterligare relevant litteratur. Stoppkriteriet för sökning av litteratur var att utvalda databaser genomsökts noggrant med utvalda nyckelord, och där ytterligare sökning inte resulterar i ytterligare relevant litteratur. Sökresultaten ansågs ej relevanta om inget nytt framgår, eller då den endast refererar till litteratur som redan täckts in i litteraturstudien.

Det tillvägagångssätt som användes för analysen av kriteriernas relevans, var att först analyserades den litteratur som kriterier utvunnits från för delmål 1. Det gjordes för att se om någon motivering ges kring varför ett visst kriterium anses relevant för automatiserade GUI-testningsverktyg. Därefter gjorde en analys utifrån en abstrakt testprocess, som valdes ut i från litteraturen. Det gjordes med syftet att ha något att förhålla sig till vid analysen, samt för att få en ökad förståelse för hur det ett kriterium avser kan vara relevant i en organisations testningsarbete.

4.2 Fallstudiens tillvägagångssätt

Fallstudiens utfördes på ett företag, med de huvudsakliga syftena att identifiera ytterligare kriterier för utvärdering (delmål 1), samt att utvärdera **ett urval** av identifierade kriterier (delmål 3). Ett urval gjordes dels av tidsmässiga skäl, men även för att det speglar hur det kan se ut i praktiken. Alla kriterier är inte relevanta i alla situationer, det beror exempelvis på vilken typ av GUI-testningsverktyg som avses.

Fallstudien utfördes på ett företag som utvecklar olika typer av GUI-applikationer. Företaget lägger i nuläget mycket tid på manuell regressionstestning, och de hade därav uttryckt ett intresse av automatiserad GUI-testning. Fallstudien inleddes med att utvecklingschefen på företaget intervjuades. Den här intervjun syftar till att få bättre kunskap om hur företaget jobbar med testning och vilka förutsättningar och behov de har kring testautomationen. Dessutom bidrar intervjun till en ökad kunskap kring hur en person i näringslivet kan se på testautomation och vad som kan vara viktigt hos ett verktyg för automatiserad GUI-testning. Den här intervjun följdes upp med ytterligare intervjuer, där personer med andra roller på företaget intervjuades. Syftet med dessa intervjuer är att de ska ge ytterligare kunskap kring företagets testningsarbete, samt vad dessa personer anser är viktigt hos ett sådant verktyg. Intervjuerna utfördes med en kvalitativ ansats, för att kunna ställa öppna frågor där den som svarar kan motivera sina svar. I utforskande syfte, där svaren inte är givna, är kvalitativa frågor att föredra över kvantitativa frågor.

När intervjuerna hade genomförts så gjordes en sammanställning av de kriterier som identifierats. En rangordning gjorde sedan bland den totala mängden kriterier, vilket gjordes i samspråk med en företagsrepresentant. När rangordningen gjorts så utfördes en utvärdering av utvalda kriterier på en av företagets webbapplikationer, för vilken ett antal tester automatiserad utifrån en testspecifikation. Utvärderingen gjordes för två Capture & Replay verktyg, då det är den typ av verktyg som passar bäst in på företagets behov. De kriterier som valdes är därför kriterier som är relevant för den typen av verktyg, exempelvis är design av testfall irrelevant i sammanhanget. Vid utvärderingen användes kriterierna enligt beskrivna sätt från litteraturen. En sammanställning av resultaten för kriterierna gjordes sedan, och ytterligare kriterier som observerades under utvärderingen lades till. Relevansen hos de ytterligare kriterier som identifierades analyserades sedan på samma sätt som beskrivet i 4.1.

5 Relaterade arbeten

Den litteratur som är mest relaterad till det här arbetet är den där utvärderingar av programvara beskrivs, särskilt sådan som berör GUI-testningsverktyg. Ytterligare är litteratur som handlar om utvärderingsmetoder, GUI-testning och testautomation relevant.

Det finns ett par arbeten kring utvärderingar av programvara, som är nära relaterade till det här arbetet. Illes m.fl. (2005) har i sin artikel definierat kriterier för utvärdering av testverktyg. De har dels specificerat kvalitetskriterier för testverktyg genom att utgå från kvalitetsstandarden ISO/IEC 9126 med utökade kriterier för leverantörskvalifikationer, leverantörssupport, samt licensiering och pris för verktyget. Illes m.fl. (2005) har även tagit fram funktionella utvärderingskriterier för testverktyg, där kriterierna delats in efter vilken del i en testprocess de berör. Deras arbete är relaterat till det här på så sätt att de också tagit fram kriterier för utvärdering av testverktyg. Skillnaden mot det här arbetet är att deras arbete inte är riktat mot någon specifik typ av testverktyg, medan det här riktar sig mot GUI-testningsverktyg. Likheter är att i det här arbetet har kriterier för urval av verktyg (selektionskriterier) delats in enligt ISO/IEC 9126 med de utökade kategorier som Illes m.fl. (2005) redogör för. Dessutom har utvärderingskriterier delats in enligt en abstrakt testprocess precis som i deras arbete.

Börjesson och Feldts (2012) arbete är ett annat arbete som är relaterat till det här. De har utvärderat två GUI-testningsverktyg som använder bildigenkänning i en industriell miljö. De har använt delar av de kriterier som Illes m.fl (2005) tagit upp, med ytterligare kriterier som de själva identifierat som är mer förknippade med deras applikationsdomän. Deras arbete påminner om det här på så sätt att de utfört en utvärdering av två GUI-testningsverktyg. De utvärderingskriterierna har därför varit intressanta för det här arbetet. Syftet med deras arbete skiljer sig dock åt mot det här. Syftet med deras arbete är att jämföra två GUI-testningsverktyg i en industriell miljö, medan det här arbetet syftar till att identifiera kriterier som kan användas vid en utvärdering av sådana verktyg. De har dessutom avgränsat sig till verktyg som använder bildigenkänning, medan det här arbetet inkluderar alla typer av GUI-testningsverktyg. Många av de utvärderingskriterier som presenteras i det här arbetet kommer från Börjesson och Feldt (2012), samt Illes m.fl. (2005), vilka fungerat som utmärkta källor.

Det arbete Behkamal m.fl. (2009) har gjort är också relaterat till det. Deras arbete handlar om hur B2B-applikationer kan utvärderas. De har analyserat olika sätt att se på programvarukvalitet (kvalitetsmodeller etc.), och valde att basera sin utvärderingsmetod på ISO/IEC 9126. Deras arbete är relaterat till det här på så sätt att de utgått från ISO/IEC 9126 när de tagit fram utvärderingskriterier. Vissa av de kriterier som tas upp i deras arbete är även relevanta som selektionskriterier i det här arbetet.

6 Kriterier

Litteraturstudien och de intervjuer som utfördes syftar till att identifiera kriterier som kan användas vid utvärdering av GUI-testningsverktyg. De kriterierna kan som Poston och Sexton (1992) redogör för, delas in i två kategorier: selektionskriterier och utvärderingskriterier. Selektionskriteriernas syfte är att användas för att göra ett urval bland den totala mängden verktyg, för att resultera i en rimlig mängd av verktyg att utvärdera. Utvärderingskriterier anger vad som ska utvärderas vid utvärderingen.

6.1 Selektionskriterier

Selektionskriterierna används som sagt för att göra en gallring bland den totala mängden verktyg. Vilka kriterier som används har en betydande roll; för slappa kriterier kan resultera i en otillräcklig gallring, medan för strikta kriterier kan resultera i att för många verktyg gallras bort. Selektionskriterierna kan därför behöva bearbetas flera gånger (Poston & Sexton, 1992). Det ska även påpekas att selektionskriterier kan vara utvärderingskriterier i andra sammanhang och tvärtom. Vanligtvis brukar selektionskriterier vara mer övergripande än utvärderingskriterier; de bör kunna besvaras utan att använda verktygen.

Kvalitet är viktigt hos programvara. I det här arbetet presenteras selektionskriterier enligt ISO/IEC 9126 då det är en mångsidig standar som är relevant för all typ av programvara (se 2.5). En utökning görs enligt Illes m.fl (2005) med kriterier för leverantörskvalifikationer, leverantörssupport, samt licensiering och pris. Det här utgör viktiga aspekter när ett verktyg ska införskaffas, exempelvis kan det finnas en viss budget för verktyget.

6.1.1 Funktionalitet

Funktionella kriterier är krav som ställs på funktionaliteten hos ett verktyg (Behkamal, 2009). Funktionella kriterier är de kanske viktigaste selektionskriterierna, då ett verktygs funktionalitet begränsar vad som går att göra. Funktionella kriterierna blir väldigt specifika för den typ av verktyg som avses. I Tabell 2 presenteras ett antal av de mest viktiga selektionskriterier kring funktionaliteten hos ett verktyg.

Tabell 2. Selektionskriterier – Funktionalitet.

Kriterium	**Kommentar**	**Referens**
Testaktiviteter det finns stöd för	Vilka testaktiviteter som ska kunna utföras utgör en fundamental fråga (testexekvering, testdesign etc.)	Lewis (2009, s.411)
Typ av GUI:n	Vilka typer av GUI-applikationer verktyget kan testa är centralt (webb, mobil etc.)	Lewis (2009, s.524)
Tillvägagångssätt för konstruktion av testscript	Vilka sätt testscript kan skapas på (exempelvis spela in användarsession).	Börjesson & Feldt (2012)
Teknik för identifiering av GUI-komponenter.	Vilken teknik verktyget använder för att identifiera GUI-komponenter.	

Verktyg kan rikta sig mot specifika testaktiviteter och typer av GUI:n vilket gör att det är viktigt att utvärdera. Vilka tillvägagångssätt som kan användas för att skapa testscript påverkar vilken kunskap som krävs för att använda verktyget, och identifieringsteknik påverkar hur känsliga testscripten är för ändringar i GUI:t.

6.1.2 Pålitlighet

Kvalitetskriteriet pålitlighet handlar om hur väl det går att lita på ett program - att det gör vad det ska och att saker inte går fel, i en viss miljö och under en viss tid (Behkamal m.fl., 2009). Till exempel om ett verktyg har en beräknad pålitlighet på 99,9% så kan man räkna med att ett fel kommer inträffa per 1000 körtimmar. Fel i det här fallet innebär att verktyget inte uppfyller de krav som ställs på det (Pressman, 2010, s.442). För att kunna uttala sig om pålitligheten hos ett verktyg krävs det således statistisk kring körningar (vilket kan göra det olämpligt att använda pålitlighetskriterier som selektionskriterier). Pålitlighet kan även bedömas genom att läsa dokumentation, samt fråga folk om deras erfarenheter kring verktyget. Pålitlighet handlar förutom frekvensen för fel, även om vilken feltolerans ett verktyg har, samt möjligheten att återhämta förlorad data vid fel (Behkamal m.fl., 2009). I Tabell 3 presenteras selektionskriterier kring pålitligheten för ett verktyg.

Tabell 3. Selektionskriterier – Pålitlighet.

Kriterium	Kommentar	Referens
Felhantering	Verktyget bör ha någon typ av felhantering vid oväntade händelser.	Lewis (2009, s.337)
Återhämtbarhet	Data bör kunna återskapas när ett system eller delar av ett system kraschar.	Behkamal m.fl. (2009)

6.1.3 Användbarhet

Användbarhet handlar om hur enkelt det är att använda ett program vilket följande attribut anger: förståbarhet, lärbarhet, samt manövrerbarhet (Behkamal, 2009). Förståbarhet handlar om hur enkelt det är att förstå ett verktyg, till exempel om en användare vill utföra X, kan användaren då göra det utan att läsa instruktioner? Lärbarhet handlar om hur enkelt det är att lära sig att använda ett verktyg. Manövrerbarhet handlar om hur enkelt det är att använda ett verktyg. Måste användare anstränga sig eller går det enkelt? I Tabell 4 visas selektionskriterier som berör användbarheten hos ett verktyg.

Tabell 4. Selektionskriterier – Användbarhet.

Kriterium	Kommentar	Referens
Lärbarhet	Ett verktyg bör vara enkelt att lära sig. Det bör till exempel finnas dokumentation och demos t.ex.	Pressman (2010, s.359)
Manövrerbarhet	Verktyget bör ha stöd för personliga inställningar för att underlätta användningen.	Behkamal m.fl. (2009)
Standarder	Följer verktyget någon användbarhetsstandard?	
Layout	Är verktygets gränssnitt inställningsbart? Verktygets gränssnitt bör vara tilltalande och bör kunna användas av personer med funktionsnedsättningar (färgblindhet etc.)	Pressman (2010, s.541)

Användbarhet kan vara svårt att kvantifiera, eftersom hur användbarhetskriterier upplevs kan variera mellan användare. Pressman (2010, s. 404) konstaterar att ISO/IEC 9126 fungerar som en utmärkt checklista för att utvärdera kvaliteten hos ett system, även om en del kvalitetskriterier (däribland användbarhet) kan vara svåra att mäta på ett tillförlitligt sätt.

6.1.4 Resursutnyttjande

Resursutnyttjande handlar om huruvida ett verktyg använder systemresurser på ett optimalt sätt, vilket anges av attributen tidsutnyttjande och resursutnyttjade (Behkamal, 2009).

Användning av diskutrymme, minne, och nätverksresurser är faktorer som påverkar ett verktygs resursutnyttjande. Ett verktyg bör även vara tidseffektivt; det ska inte ta verktyget flera timmar att utföra en simpel uppgift. Resursutnyttjande är viktigt att beakta för att verktyget ska gå att använda i avsedd miljö, samt att saker utförs tillfredställande snabbt. I Tabell 5 presenteras selektionskriterier som berör resursutnyttjande hos ett verktyg.

Tabell 5. Selektionskriterier – Resursutnyttjande.

Kriterium	**Kommentar**	**Referens**
Systemkrav	Vilka systemkrav ett verktyg har, vilket bör framgå i specifikationen för verktyget.	Behkamal m.fl. (2009)
Tidseffektivitet	Hur tidseffektivt verktyget utför något kan vara viktigt. Till exempel kan ett simpelt scenario utföras och en tidsmätning göras över hur lång tid det tar.	Börjesson & Feldt (2012)

6.1.5 Underhållsbarhet

Underhållsbarhet handlar om huruvida ett verktyg med enkelhet kan lagas och utvecklas, vilket följande attribut anger: analyserbarhet, förändringsbarhet, stabilitet, och testbarhet (Behkamal, 2009). För god analyserbarhet är det viktigt att det finns dokumentation kring verktyget, det är även bra om källkoden finns tillgänglig och att den är begriplig (Heitlager, Kuipers & Visser, 2007). Förändringsbarhet kan till exempel handla om huruvida källkoden är tillgänglig, för att ändringar ska kunna göras manuellt. Det är enklare att ändra ett verktyg som är open source eller byggt på egen hand jämfört med ett kommersiellt verktyg, där källkoden oftast inte finns tillgänglig. Stabilitet handlar om hur känsligt ett system är för ändringar, d.v.s. hur stor risken är för att något ”dåligt” sker vid ändringar. Testbarhet handlar om hur enkelt det är att testa ett verktyg vid ändringar. I Tabell 6 presenteras kriterier kring underhållsbarheten för ett verktyg.

Tabell 6. Selektionskriterier – Underhållsbarhet.

Kriterium	**Kommentar**	**Referens**
Dokumentation	Verktyget bör vara väldokumenterar för att ge en förbättrad analyserbarhet.	Heitlager m.fl (2007)
Utformning av kod	Hur koden ser ut – dess volym, komplexitet och läsbarhet påverkar underhållsbarheten.	
Tillgänglig källkod	För att verktyget ska kunna editeras med enkelhet är det bra om källkoden finns tillgänglig. Typ av programvara blir därför relevant (open source, kommersiellt, eller bygga-själv).	Hass (2008, s.365)

För att kunna veta hur väl det går att utföra underhållsarbete för ett verktyg krävs det kännedom om dess dokumentation och kod. Det kan vara bra att reflektera över om det är viktigt med tillgänglig källkod, vilket i stor utsträckning påverkar valet av verktyg. Hur koden ser ut påverkar också underhållsbarheten för verktyget.

6.1.6 Portbarhet

Portbarhet handlar om hur enkelt ett verktyg kan flyttas från en miljö till en annan, vilket följande attribut anger: anpassningsbarhet, installerbarhet, överensstämmelse, samt ersättningsbarhet. Anpassningsbarhet handlar om hur väl verktyget kan anpassa sig till olika miljöer, till exempel hur det fungerar i olika operativsystem. Installerbarhet handlar om hur

mycket ansträngning som krävs för att installera verktyget. Överensstämmelse handlar om hur väl olika delar av verktyget går att överföra till andra situationer, till exempel om den databas som används kan användas för andra miljöer också. Ersättningsbarhet handlar om hur olika komponenter för verktyget går att ersätta. Till exempel om verktyget levereras med en viss typ av databas, då handlar ersättningsbarhet om den går att byta ut mot en annan. I Tabell 7 presenteras selektionskriterier som berör portbarheten hos ett verktyg.

Tabell 7. Selektionskriterier - Portbarhet.

Kriterium	Kommentar	Referens
Plattformsstöd	Vilka plattformar ett verktyg fungerar på.	Poston & Sexton (1992)
Integration med andra verktyg	Att verktyget kan samverka med andra verktyg som används i utvecklingen är ofta önskvärt. Exempelvis ett verktyg för versionshantering.	
Enkelt att installera	Hur enkelt är verktyget att installera? Finns det en installationsguide eller måste det skötas manuellt?	Behkamal m.fl. (2009)
Ersättningsbarhet	Hur väl det går att byta ut olika komponenter för verktyget? Går det till exempel att byta IDE?	

6.1.7 Leverantörskvalifikationer

Leverantörskvalifikationer, leverantörssupport (5.1.8), samt licensiering och pris (5.1.9) är inte kvalitetskriterier enligt ISO/IEC 9126, men är ändå bra att ha med i utvärderingen. Leverantörskvalifikationer handlar exempelvis om hur välkänd leverantören är och hur stor marknadsandel leverantören har m.m. (Illes m.fl., 2005). I Tabell 8 presenteras selektionskriterier som berör kvalifikationer för leverantören av ett verktyg.

Tabell 8. Selektionskriterier - Leverantörskvalifikationer

Kriterium	Kommentar	Referens
Tidigare erfarenheter av leverantören	Det är bra att reflektera över tidigare erfarenheter man har av en leverantör. Det kan också vara bra att fråga andra.	Carvallo & Franch (2006)
Marknadsandel	Vilka verktyg andra använder kan indikera på vilka som är bäst, det behöver nödvändigtvis inte vara så dock.	
Utmärkelser och certifikat	Har ett verktyg fått utmärkelser eller erhållit certifikat kan det indikera på att verktyget är bra.	

6.1.8 Leverantörssupport

Vilken support en leverantör erbjuder kan vara viktigt när problem uppstår. För verktyg som kostar pengar kan man förvänta sig att få en viss support när problem eller frågor dyker upp. När det gäller open source-verktyg går det dock inte att förvänta sig samma support, eftersom verktyget är gratis. Det brukar dock finnas diskussionsforum för många open source-verktyg i vilka frågor kan ställas, så i vissa fall kan ändå hjälp fås. I Tabell 9 presenteras selektionskriterier som berör leverantörssupport för ett verktyg.

Tabell 9. Selektionskriterier – Leverantörssupport.

Kriterium	Kommentar	Referens
Erbjuden support	Framgår det vilken support som erbjuds kring verktyget då problem eller frågor uppstår?	Carvallo & Franch (2006)

Träningsmöjligheter	Erbjuds träning för verktyget? Det kan vara värdefullt i samband med implementation av verktyget inom en organisation.	

6.1.9 Licensiering och pris

De här kriterierna handlar om licensiering och pris för ett verktyg. De här kriterierna är viktiga, och i vissa fall är kanske priset det första som ses över om en viss avsatt budget finns för verktyget. I Tabell 10 presenteras kriterier som berör licensiering och pris för ett verktyg.

Tabell 10. Selektionskriterier - Licensiering och pris.

Kriterium	**Kommentar**	**Referens**
Pris och licensiering	Pris och licensieringsalternativ för verktyget bör ses över.	Illes m.fl. (2005)
Garantier	Vilka garantier som ges för verktyget är bra att se över. Ingår eventuella uppdateringar/tillägg?	Carvallo & Franch (2006)
Ägandeskap	Hur ägandeskapet ser ut, vilka rättigheter som finns kring produkten exempelvis, är bra att känna till för att undvika otillåten distribution etc.	

Det finns många olika prisnivåer för GUI-testningsverktyg. Det är därför viktigt att reflektera över tillgänglig budget som finns kring verktyget. Är det tänkt att verktyget ska användas på flera maskiner är det även viktigt att se över vilka möjligheter som finns kring det. Det kan behövas speciella licenser för det etc.

6.2 Utvärderingskriterier

När ett urval av kandidatverktyg har valts ut och en utvärdering ska göras för dessa krävs det kriterier som anger vad som ska utvärderas. Hur kriterier för utvärdering ser ut och kategoriseras kan göras på olika sätt (se 2.5). Här görs en indelning utifrån vilken del i testprocessen (se 2.2.4) kriterierna berör, en indelning som förespråkas av Illes m.fl. (2005).

6.2.1 Testplanering och översikt

Testaktiviteter upptar ofta mycket av den totala tiden inom utvecklingsprojekt (Grindal m.fl, 2006.) Det är därför viktigt att planera för testning och övervaka den. En viktig del kring testplaneringen är att den organisatoriska testprocessen ska anpassas till ett visst projekt vilket bland annat omfattar att testaktiviteter ska prioriteras och resurser för testningen ska fördelas (Illes m.fl., 2005). Nyckelaktiviteter för övervakning är bland annat att möjliggöra projektöversikt, anpassa testplanen efter rådande omständigheter, samt koordinera testningsaktiviteter. I Tabell 11 presenteras kriterier relaterade till testplanering och översikt.

Tabell 11. Utvärderingskriterier – Testplanering och översikt.

Utvärderingskriterium för verktyg	**Referens**
1. Verktyget kan användas för att anpassa den organisatoriska testprocessen.	Illes m.fl. (2005)
2. Verktyget förser stöd för applikationsspecifika karaktäristiker, som kräver speciella testtekniker (exempelvis model-driven testing).	
3. Verktyget kan användas för att testa speciella applikationsdomäner (exempelvis för webbapplikationer).	

4. Verktyget kan användas för planering av testprocessen (schemaläggning, projektöversikt, riskhantering).	
5. Verktyget ger stöd i att kontrollera testaktiviteter, genom att: • ge en översikt över förväntad och faktisk exekveringstid per testfall • erbjuda täckningsmått för att mäta framstegen för testaktiviteter	
6. Viktiga villkor kan anges i verktyget, deadlines för testfall etc.	Hass (2008, s.43)
7. Verktyget kan ange strukturen för testspecifikationer.	

6.2.2 Analys och design

När testningen har planerats, så ska en analys göras och testfall designas. En analys bör göras kring vilka testfall som kan vara viktiga, och hur dessa testfall kan förbättras. När det gäller design av testfall, så ska testfall väljas utifrån den teststrategi som definierats, testvillkor ska tas fram, och logiska testfall ska tas fram och dokumenteras (Illes m.fl., 2005). I Tabell 12 presenteras kriterier relaterade till analys och design av testfall för verktyget.

Tabell 12. Utvärderingskriterier – Analys och design av testfall.

Utvärderingskriterium för verktyg	**Referens**
1. Det går m.h.a. verktyget att designa testfall för krävd testnivå (enhetstestning, systemtestning etc.).	Illes m.fl. (2005)
2. Testteknik kan anges för verktyget.	
3. Testvillkor kan anges utifrån vald testteknik för verktyget.	
4. Informationen för testfall kan struktureras m.h.a. fördefinierade mallar i verktyget.	
5. Högnivå testfall kan genereras utifrån modeller m.h.a. verktyget.	
6. Design av testfall för att testa kvalitetskriterier hos applikationen (t.ex. load-testing) kan göras m.h.a. verktyget.	
7. Input för testfall kan genereras m.h.a. verktyget via input-modeller.	Hass (2008, s.373)
8. Högnivå testfall kan genereras av verktyget utifrån källkod.	
9. Förväntad output för testfall kan uppskattas av verktyget.	Memon m.fl. (2003)
10. Verktyget kan automatiskt skapa en modell som efterliknar GUI:t, genom att analysera dess källkod.	
11. Verktyget kan avgöra om ett existerande testfall är användbart eller oanvändbart för en modifierad version av GUI:t.	Memon (2008)
12. Verktyget kan automatiskt reparera oanvändbara testfall.	

6.2.3 Implementation av automatiseringen

Utifrån logiska testfall ska konkreta testfall och testdata tas fram och implementeras (Illes m.fl., 2005). När det gäller testautomatisering ska testscript tas fram i det här skedet. I Tabell 13 presenteras kriterier som berör testimplementation.

Tabell 13. Utvärderingskriterier - testimplementation.

Utvärderingskriterium för verktyg	**Referens**
1. Verktyget möjliggör inspelning av exekverbara testfall.	Illes m.fl. (2005)
2. Testscript går att editera med verktyget.	
3. Konkreta testfall kan genereras av verktyget utifrån modeller.	
4. Verktyget möjliggör generering av stubs, drivers och mock-objekts.	

5. Verktyget har stöd för keyword-driven testing.	Hass (2008, s.377)
6. Verktyget har stöd för data-driven testing.	Lewis (2009, s. 338)
7. Antal rader kod för genererade script.	Börjesson m.fl. (2012)
8. Verktyget har bildigenkänning i IDE:n.	
9. Verktyget möjliggör inspelning av tester på en fjärrdator.	
10. Verktyget har testsvit support.	
11. Dynamiska GUI-komponenter kan hanteras av verktyget.	
12. Tid att konstruera testscript.	

6.2.4 Testexekvering

Testexekvering utgör en central del hos ett verktyg för automatiserad testning. Att automatisera exekveringen av de testscript som utvecklats är väsentligt för att automationen ska vara meningsfull. I Tabell 14 presenteras utvärderingskriterier kring testexekvering.

Tabell 14. Utvärderingskriterier – Testexekvering.

Utvärderingskriterium för verktyg	Referens
1. Verktyget har stöd för att exekveringen av ett testfall kan pausas och sedan återupptas.	Illes m.fl. (2005)
2. Rangordning över testfall kan sättas m.h.a. verktyget.	
3. Verktyget möjliggör rollback till initialt läge, om testfall misslyckas.	
4. Verktyget möjliggör exekvering av: inspelade tester, inspelade och editerade tester, samt tester som skapats manuellt.	
5. Verktyget möjliggör exekvering av inspelade tester för att utvärdera kvalitetskriterier (t.ex. belastningstestning).	
6. Uppspelningstid för testscript (tiden det tar att exekvera de testfall som automatiserats i testscript).	Börjesson & Feldt (2012)
7. Verktyget kan ge realtidsfeedback vid scriptexekvering.	
8. Applikationsspecifika karaktäristiker kan testas.	

6.2.5 Testexekvering - fånga och jämföra resultat

När testfall exekveras ska erhållet resultat jämföras med förväntat resultat, data bör även samlas in automatiskt över exekverade teststeg och vilka resultat som erhållits för att underlätta i analysen av testningen (som också kan automatiseras). Det här steget är starkt förknippat med testorakel, som Xie och Memon (2006) redogör för, och som togs upp i 2.2.3. I Tabell 15 presenteras utvärderingskriterier kring fånga och jämföra resultat.

Tabell 15. Utvärderingskriterier - Fånga och jämföra resultat.

Utvärderingskriterium för verktyg	Referens
1. Verktyget kan samla in logginformation för exekverade tester.	Illes m.fl. (2005)
2. Verktyget kan jämföra faktisk output med förväntad output för ett test.	
3. Testorakel kan skapas m.h.a. verktyget.	Hass (2008, s.374)

6.2.6 Testrapportering

Testrapportering är en viktig del i testprocessen för att det ska gå att få en spårbarhet kring vilka tester som körts och dess utfall. Testrapportering är särskilt viktigt vid automatiserad

testning, där det vanligtvis inte finns någon som observerar testexekveringen. I Tabell 16 presenteras utvärderingskriterier som berör testrapportering.

Tabell 16. Utvärderingskriterier – Testrapportering.

Utvärderingskriterium för verktyg	**Referens**
1. Verktyget kan sammanställa logginformation för exekverade testfall.	Illes m.fl. (2005)
2. Verktyget möjliggör att en inställningsbar mängd testdata *sparas* till en testrapport.	
3. Verktyget möjliggör att en inställningsbar mängd testdata *visas* för en testrapport.	
4. Testrapporter kan automatiskt skickas till en angiven e-mail.	Memon m.fl. (2003)

6.2.7 Debuggingstöd

Debugging är ingen testaktivitet (Hass, 2008, s. 371). Det är dock viktigt att verktyg förser debuggingstöd, eftersom när testningen har utförts vill man ofta spåra och åtgärda eventuella buggar. Det är därför viktigt att kunna få en spårbarhet över problem från testkörningar. I Tabell 17 presenteras utvärderingskriterier kring debuggingstöd.

Tabell 17. Utvärderingskriterier – Debuggingstöd.

Utvärderingskriterium för verktyg	**Referens**
1. Verktyget ger stöd för att specificera problem/fel genom att förse användaren med fördefinierade mallar.	Illes m.fl. (2005)
2. Verktyget kan generera noteringar vid fel.	
3. Fel kan prioriteras m.h.a. verktyget.	
4. Verktyget möjliggör övervakning av ändringar på efterfrågningar/felnoteringar och dess nuvarande status.	
5. Verktyget kan ge statistisk information över körningar.	

6.2.8 Underhållsstöd

Underhållsstöd är ingen testaktivitet men det är viktigt att verktyget förser stöd för underhåll av den automatiserade testningen för att automationen ska bli effektiv. Underhållsstöd omfattar bland annat hantering av tester (versionshantering, lagring, delning m.m.), att det går att få en spårbarhet mellan olika delar av testmiljön (krav - konkreta testfall etc.), att ändringar på testobjekt går att spåra med mera (Illes m.fl., 2005). Underhållsstöd för skötsel av tester och testdata är kritiskt, för att underhållsarbetet inte ska bli ohanterligt. I Tabell 18 presenteras kriterier kring underhållsstöd av automationen.

Tabell 18. Utvärderingskriterier – Underhållsstöd.

Utvärderingskriterium för verktyg	**Referens**
1. Verktyget förser stöd för skötsel av tester (versionshantering etc.)	Illes m.fl. (2005)
2. Verktyget kan bidra till att en spårbarhet fås mellan olika element hos testningen, (t.ex. mellan logiska och konkreta testfall).	
3. Verktyget kan ge en spårbarhet över ändringar för testobjekt (testscript etc.) och kommunicera dessa.	
4. Verktyget förser stöd vid underhåll av testdata för testfall.	
5. Verktyget har snapshots-inrättningar.	

7 Fallstudie

En fallstudie utfördes med syftet att identifiera ytterligare relevanta kriterier för utvärdering av automatiserade GUI-testningsverktyg, samt för att utvärdera ett urval av för fallstudien relevanta kriterier. Syftet med utvärderingen är att se om, och hur de här kriterierna kan användas i praktiken, och om de ger värdefull information vid en utvärdering. Se kapitel 3 för mer information om fallstudiens syfte. Fallstudien inleddes med en intervju med utvecklingschefen på företaget för fallstudien. Den här personen hade uttryckt att företaget är intresserade av att kunna ha automatiserad GUI-testning, då företaget lägger mycket tid på manuell testning av GUI:n i nuläget. Intervjun utfördes därför med syftet att få mer kunskap om vilka behov företaget har och på vilket sätt automatiserad GUI-testning skulle kunna vara till nytta för företaget. En utvärdering bör som nämnt inledas med någon form av behovsanalys (Poston & Sexton, 1992; Hass, 2008, s. 364). Den här intervjun kan ses som en behovsanalys i liten skala. Den intervjuguide som användes för den här intervjun finns att ta del av i Appendix A.

Utifrån den inledande intervjun och genom den litteraturstudie som gjorts, så kunde ett antal selektionskriterier upprättas för användning i fallstudien. De här selektionskriterierna sattes upp i samspråk med intervjupersonen. Den inledande intervjun kompletterades sedan med att ytterligare tre personer intervjuades. De här personerna har andra roller på företaget och syftet med de här intervjuerna är att få flera perspektiv på hur företaget jobbar med testning, och att kunna identifiera kriterier för utvärdering.

När intervjuerna utförts sattes de kriterier som identifierats upp, som inte redan täckts in av litteraturstudien. Därefter valdes verktyg ut utifrån uppsatta selektionskriterier, genom att en sammanvägning gjordes över hur väl de identifierade verktygen uppfyllde kriterierna. En utvärdering gjordes sedan för utvalda verktyg på en webbapplikation som företaget utvecklat. Vissa av de kriterier som utvärderades krävde att verktygen användes och att testfall kördes, medan för vissa räckte det att läsa dokumentation. Ett antal testfall för applikationen automatiserades och kördes, vilket gjordes utifrån en testspecifikation.

7.1 Intervjuerna

Intervjun med utvecklingschefen på företaget följdes upp med ytterligare tre intervjuer. De här intervjuerna utfördes med syftet att få ett bredare perspektiv av hur företaget jobbar med testning och vilka behov de har, samt att kunna identifiera kriterier för utvärdering. Intervjuguiden som användes för de här intervjuerna finns i Appendix B.

Den första personen som intervjuades är avdelningschef för kundstöd. Den här rollen innebär mycket kundkontakt. När fel rapporteras av kunder är det personerna som jobbar med kundstöd som ska undersöka att det som rapporteras som fel faktiskt är fel. Det innebär att användarscenarion måste upprepas. Personen utför även övergripande testning av hela system med syftet att reda ut hur saker fungerar och utifrån det kunna komma med förslag på ändringar. Dessutom behöver de som jobbar med kundstöd ha övergripande kunskap om de olika systemen, för att kunna ge bra support till kunderna. Personen säger sig lägga ungefär fem timmar i veckan på att utföra testning. Personen har intervjuats om alla kategorierna (testning generellt, automatiserad testning, och testverktyg).

Den andra personen som intervjuades jobbar med kundstöd och utbildning. Den här rollen kräver att personen kan de olika systemen för att kunna hjälpa kunderna, båda i utbildningssyfte och när kunderna upplever problem. I sitt arbete lägger personen uppskattningsvis 20% av tiden på testning. Personen har intervjuats om kategorierna testning generellt och automatiserad testning.

Den tredje personen som intervjuades jobbar med projektledning, kravhantering, och är även testledare. Den här rollen innebär att personen har ett övergripande ansvar över testningsarbetet, vilket kräver vetskap om vad som ska testats, vem som ska testa vad osv. Förutom att utföra testning är personen även med och tar fram testspecifikationer tillsammans med utvecklare. Personen säger sig lägga mycket av sin arbetstid på testning, vissa veckor upp till 70%. Personen har intervjuats om alla kategorierna.

Vilka uppgifter olika personer har kring testningen på företaget beror på deras arbetsroll. Gemensamt bland de som intervjuats är att de poängterar vikten av kommunikation mellan avdelningar kring testningsarbetet. Driftavdelningen ska förse de som testar med en lämplig testningsmiljö, utvecklingsavdelningen ska skriva testspecifikationer och utföra testning, och kundstödsavdelningen ska rapportera fel som kunderna upplevt till berörda utvecklare. Dessutom finns det särskilda testledare som har ansvar för att testning blir utfört. Dokumentation av testning sker i testspecifikationen, medan buggar dokumenteras i ett specifikt dokument. De buggar som upptäcks vid testning rapporteras till berörd utvecklare.

Två typer av testning nämndes i intervjuerna: regressionstestning och acceptanstestning. Regressionstestning är återkommande testning som brukar utföras vid större ändringar i system för att verifiera att helheten fortfarande fungerar. Acceptanstester är tester som är specifik för ny funktionalitet, och dess syfte är att verifiera att den nya funktionaliteten uppfyller användarkraven. Det framgår även att testning huvudsakligen utförs utifrån testspecifikationer, men att ytterligare testning brukar ske utifrån personliga erfarenheter. När personerna frågades om vad i företagets testningsarbete de ansåg kunde förbättras gavs lite olika svar. En av de saker som togs upp var att kommunikation mellan avdelningar kunde förbättras kring testningsarbetet. En annan sak som togs upp var att testautomation skulle vara bra för att kunna få bort en del av de tråkiga återkommande testerna som måste utföras.

När det gäller frågorna kring testautomation nämndes det att testautomation är något som företaget funderat över, och att de i nuläget har ett antal automatiserade tester i verktyget WatiN. Det fanns dock en osäkerhet kring hur aktuella de här testerna är, och efter intervjuerna visade det sig att de inte längre är aktuella. Att testerna inte hållits uppdaterade ansågs framförallt bero på att verktyget inte givit något bra stöd för att underhålla testscripten. De fördelar som nämndes med testautomation var tidsbesparingar, mer och frekventare testning, samt att testning kan utföras oberoende av kontorstider. En annan aspekt som togs upp, var att annan typ av testning kan utföras genom automatiserad testning, till exempel belastningstestning. Risker som nämndes var bland annat att för stor tilltro ges till automationen, och att det kan leda till en oklarhet kring vad som blivit testat.

Två av personerna intervjuades om testverktyg. En av dem hade lite erfarenhet av sådana verktyg, men ville inte uttala sig i detalj om det, eftersom det var längesedan personen använde ett sådant verktyg. De egenskaper personerna angav att de eftertraktar hos ett verktyg för testautomatisering är bland annat att verktyget ska kunna användas av personer som inte har utvecklarkunskaper och att delar av script ska kunna editeras men att helheten ändå består. Verktyget ska även ge tydliga felmeddelanden. Ytterligare nämndes

förutsägbarhet som ett viktigt kriterium, d.v.s. att samma resultat ges vid exekvering varje gång.

Intervjuerna gav framförallt mer kunskap om hur företaget jobbar med testning, samt vilka behov de har kring testautomation. Intervjuerna resulterade även i ett antal utvärderingskriterier och en förbättrad kunskap kring testning

7.2 Val av verktyg att studera

Utifrån den inledande intervjun och litteraturstudien, så sattes ett antal selektionskriterier upp vilket gjordes i samspråk med utvecklingschefen på företaget. Följande selektionskriterier sattes upp:

- **Typer av GUI:n** – Verktyget ska kunna testa webbapplikationer, eftersom det är vad företaget huvudsakligen utvecklar.
- **Testaktiviteter** – Verktyget ska kunna utföra testexekvering, då dess huvudsakliga uppgift ska vara att utföra regressionstestning. Det ska även gå att spela in testfall genom att fånga en användarsession; verktyget ska alltså ha Capture & Replay funktionalitet.
- **Identifieringsteknik av GUI-komponenter** – Verktyget ska kunna identifiera objekt på en webbsida genom DOM:n (Document Object Model) för en webbsida, till exempel med hjälp av XPath (se http://www.w3schools.com/dom/ och http://www.w3schools.com/xpath/). Anledningen till det är att det möjliggör att objekt kan identifieras unikt i de flesta fall. Bildigenkänning kan vara ett bra alternativ, men det kan vara opålitligt om det finns flera likadana objekt.
- **Integration med andra verktyg** – Testverktyget ska kunna integreras med andra verktyg som redan används i utvecklingsprocessen. De verktygen är i det här fallet Visual Studio och Team Foundation Server. Integration innebär här att det ska gå att arbeta med testscripten i Visual Studio som är en IDE, och ha versionshantering av scripten m.h.a. Team Foundation Server vilket är ett verktyg i det avseendet.
- **Tillgängligt** – Verktyget ska inte behöva köpas för att utvärderas. Det innebär att verktyget ska vara open source eller ha en pröva-på-period på minst 30 dagar.
- **Väldokumenterat** – Verktyget ska vara väldokumenterat eftersom det blir svårt att lära sig att använda verktyget annars. Den officiella dokumentationen var den första utgångspunkten, och om den ansågs bristfällig valdes verktyget bort.
- **Scriptspråk** – Det här kriteriet vägde inte särskilt tungt, men verktyg som erbjuder valmöjligheter av scriptspråk prioriterades då det underlättar användandet. Verktyg som har C# som möjligt scriptspråk prioriterades. Det gjordes eftersom det matchar de kunskaper som finns på företaget.

En sammanvägning av de här kriterierna gjordes, utifrån tillgänglig dokumentation för verktygen. De två första kriterierna gjorde att ett stort antal verktyg kunde sållas bort direkt; verktyg som inte kan utföra funktionell testning av webbapplikationer valdes bort. Det fanns fler verktyg än de som valdes ut för utvärdering som uppfyllde alla selektionskriterier. De som valdes ut var de som uppfylla kriterierna bäst, till exempel de som ansågs mest väldokumenterade. En avgränsning gjordes till två verktyg, för att inom ramarna för det här

examensarbetet hinna göra en utvärdering för dem. Följande två verktyg valdes ut för utvärdering, baserat på selektionskriterierna:

1. **Selenium** – En samling av olika open source-verktyg som förser stöd f testautomatisering för webbapplikationer (se http://docs.seleniumhq.org/). Selenium innefattar bland annat ett omfångsrikt API (Selenium Webdriver) som kan användas för att programmera automatiserade tester manuellt. Ett annat Selenium verktyg är Selenium IDE, som är ett plugin till Mozilla Firefox med Capture & Replay funktionalitet (se 2.4.1). Selenium IDE (version 2.0) uppfyller alla selektionskriterierna och valdes därför ut i det här arbetet.

2. **Ranorex** – Ett verktyg som uppfyller alla ovanstående kriterier. Det som talade emot Ranorex var att det har en kostnad. Dock kan det användas för att testa många typer av GUI-applikationer vilket är en fördel. En annan fördel är att identifiering av GUI-komponenter kan göras på flera sätt (XPath, referenser, bildigenkänning). Ranorex version 4.0 användes i det här arbetet. För mer information, se: http://ranorex.com/.

7.3 Utvärdering av utvärderingskriterier

Litteraturstudien och de intervjuer som utfördes, resulterade i en lista med utvärderingskriterier (se 6.2). Ifrån de kriterierna, gjordes ett urval av för fallstudien relevanta kriterier, vilka utvärderades för två Capture & Replay verktyg. Nedan ges en sammanställning kring det. I Appendix C ges utförligare motiveringar till svaren.

Tabell 19. Implementation av automatiseringen - utvärdering av kriterier.

Kriterium	Selenium	Ranorex
1. Verktyget möjliggör inspelning av exekverbara testfall.	Ja.	Ja.
2. Testscript går att editera med verktyget.	Ja.	Ja.
3. Konkreta testfall kan genereras av verktyget utifrån modeller.	Nej.	Nej.
4. Verktyget har stöd för keyword-driven testing.	Ja.	Ja.
5. Verktyget har stöd för data-driven testing.	Delvis.	Ja.
6. Antal rader kod för genererade script (i det här fallet kod som genererats genom inspelning av fem användarinteraktioner).	241	505
7. Verktyget har bildigenkänning i IDE:n.	Nej.	Ja.
8. Verktyget möjliggör inspelning av tester på en fjärrdator.	Delvis.	Ja.
9. Verktyget har testsvit support.	Ja.	Ja.
10. Dynamiska GUI-komponenter kan hanteras av verktyget (t.ex. drop-down listor).	Delvis.	Ja.

I Tabell 19 ovan presenteras utvärderingen av kriterier för implementation av automatiseringen (se 6.2.3). Många av de här kriterierna kan besvaras med ett ja eller nej. En del kriterier kan dock specificeras ytterligare vid behov, till exempel för kriterium 1 kan en lista sättas upp med de typer av GUI-applikationer som det går att spela in tester för. Kriterium 6 - antalet rader kod för generade script - kan ha betydelse när det gäller underhåll av script, vilket Börjesson och Feldt (2012) påpekar. När kod genereras i samband med att en användarsession spelas in med ett Capture & Replay verktyg exempelvis, så är det relevant hur mycket kod som genereras, då det påverkar i underhållssyfte. Det är viktigt att beakta att det kan finnas skillnader hos den kod som genereras. Ranorex har per default

rapporteringsfunktionalitet, vilket inte Selenium har etc. Kriterium 10 – hantering av dynamiska objekt, kan vara ett viktigt kriterium. Särskilt för webbapplikationer som kan ha drop-down listor och andra typer av objekt som laddas dynamiskt. Det här kriteriet kan utvärderas på olika sätt. I det här fallet gjordes det genom att testa hur väl verktygen kan hantera drop-down listor vid återuppspelning. Det visade sig att Selenium kräver att pauser läggs in manuellt så objekt hinner laddas.

Tabell 20. Testexekvering - utvärdering av kriterier.

Kriterium	**Selenium**	**Ranorex**
1. Exekvering av testscript för verktyget kan ske via en fjärrdator.	Nej.	Ja.
2. Verktyget har stöd för att exekveringen av ett testfall kan pausas och sedan återupptas.	Ja.	Ja.
3. Rangordning över testfall kan sättas m.h.a. verktyget.	Nej.	Nej.
4. Verktyget möjliggör rollback till initialt läge, om ett test misslyckas.	Nej.	Ja.
5. Verktyget möjliggör exekvering av: inspelade tester, inspelade och editerade tester, samt manuella tester.	Ja.	Ja.
6. Verktyget möjliggör exekvering av inspelade tester för att utvärdera kvalitetskriterier (t.ex. belastningstesting).	Nej.	Nej.
7. Verktyget kan ge realtidsfeedback vid scriptexekvering.	Ja.	Ja.
8. Applikationsspecifika karaktäristiker kan testas. I det här fallet har möjligheten att testa i olika webbläsare utvärderats. I det här fallet Internet Explorer, Mozilla Firefox, och Google Chrome.	2/3.	3/3.

I Tabell 20 ovan presenteras utvärderingen av kriterier för testexekvering (se 6.2.4). Många av kriterierna för testexekvering är av karaktären ja/nej, även om vissa går att specificera ytterligare. Kriterium 6 - om det är möjligt att utvärdera kvalitetskriterier med verktyget - kan i vissa fall vara viktigt och bör i de fallen utgöra ett selektionskriterium. Kriterium 8 beaktar att den applikationsdomän som avses kan ställa speciella krav på verktyget. Hur det här kriteriet (kriterierna) utformas kan variera i stor utsträckning. I det här fallet har möjligheten att testa i olika webbläsare undersökts.

Tabell 21. Fånga och jämföra resultat (testexekvering) - utvärdering av kriterier.

Kriterium	**Selenium**	**Ranorex**
1. Verktyget kan samla in loggningsinformation för exekverade testscript.	Ja.	Ja.
2. Verktyget kan jämföra faktisk och förväntad output.	Ja.	Ja.
3. Testorakel kan skapas m.h.a. verktyget.	Ja.	Ja.

I Tabell 21 ovan presenteras utvärderingen av kriterier för testexekvering - fånga och jämföra resultat (se 6.2.5). De kriterier som identifierats har utvärderats genom att undersöka vilka olika möjligheter som finns för respektive verktyg. Det gick att utskilja två typer av valideringstekniker: bildvalidering, och validering av egenskaper hos GUI-komponenter.

Tabell 22. Testrapportering - utvärdering av kriterier.

Kriterium	Selenium	Ranorex
1. Verktyget kan ge en sammanställning av logginformation för exekverade testfall (testrapport).	Delvis.	Ja.
2. Testrapporter kan automatiskt skickas till en angiven e-mail.	Delvis.	Delvis.
3. Verktyget möjliggör att en inställningsbar mängd testdata *sparas* till en testrapport.	Delvis.	Ja.
4. Verktyget möjliggör att en inställningsbar mängd testdata *visas* för en testrapport.	Delvis.	Ja.

I Tabell 22 ovan presenteras utvärderingen av kriterier för testrapportering (se 6.2.6). Testrapportering är en aktivitet som är särskilt viktigt för automatiserade testverktyg eftersom det inte alltid finns någon som observerar tesexekveringen. Verktyget bör kunna skapa testrapporter automatiskt som innehåller väsentlig testinformation. Det kan även vara bra att utvärdera om det går att ange vad som ska visas i testrapporterna. Många av kriterierna kan besvaras med ja/nej, men vissa går att specificera mer. Antalet olika rapporteringsmöjligheter kan jämföras (vilken information som visas), samt vilket/vilka filformat rapporterna kan sparas till.

Tabell 23. Debuggingstöd - utvärdering av kriterier.

Kriterium	Selenium	Ranorex
1. Verktyget ger stöd för att specificera problem/fel genom att förse användaren med fördefinierade mallar.	Nej.	Nej.
2. Verktyget kan generera noteringar vid fel.	Ja.	Ja.
3. Fel kan prioriteras m.h.a. verktyget.	Nej.	Nej.
4. Verktyget möjliggör övervakning av ändringar på förfrågningar/felnoteringar och dess nuvarande status.	Nej.	Nej.
5. Verktyget kan ge statistisk information över körningar.	Nej.	Nej.

I Tabell 23 ovan presenteras utvärdering av kriterier för debuggingstöd (se 6.2.7) Debuggingstöd är ingen testaktivitet, men det är ändå viktigt att verktyget förser stöd för att underlätta när debugging ska utföras. Alla de här kriterierna är av karaktären ja/nej, och det är svårt att specificera dem mer än så. Kriterium 5 skulle dock kunna specificeras mer, i form av vilken statistisk information som kan ges. Det enda debuggingstöd verktygen förser är att felnoteringar ges.

Tabell 24. Underhållsstöd - utvärdering av kriterier.

Kriterium	Selenium	Ranorex
1. Verktyget förser stöd för skötsel av testscript (versionshantering, delning etc.)	Delvis.	Ja.
2. Verktyget kan bidra till att en spårbarhet fås mellan olika element hos testningen (t.ex. mellan testkrav och testfall).	Nej.	Nej.
3. Verktyget kan ge en spårbarhet över ändringar för testobjekt (testscript etc.), och kommunicera dessa.	Delvis.	Delvis.
4. Verktyget förser stöd vid underhåll av testdata för testfall	Delvis.	Ja.
5. Verktyget har snapshots-inrättningar (d.v.s. det går att frysa testmiljön i dess nuvarande tillstånd).	Nej.	Ja.

I Tabell 24 ovan presenteras utvärderingen av kriterier för underhållsstöd (se 6.2.8) Verktygen bör förse stöd för skötsel av tester. Framförallt versionshantering och delning kan vara viktigt, då det kan finnas många tester och flera personer som behöver ta del av dem. I Selenium ges stöd för versionshantering genom att scripten kan genereras i ett format som kan hanteras av programvara för versionshantering. I Ranorex ges bättre stöd genom att det finns ett inbyggt gränssnitt mot Subversion (SVN) och Team Foundation Server (TFS). Stöd i att underhålla testdata kan vara viktigt, och det här kriteriet kan tolkas på flera sätt. En typ av underhållsstöd är att verktyget har keyword-driven och/eller data-driven testing. Selenium ger visst underhållsstöd, medan Ranorex ger bra underhållsstöd (se Appendix C).

7.3.1 Kriterier identifierade via praktiken

Här presenteras de kriterier som identifierats i praktiken, genom intervjuer och observationer, vilka inte täckts in av litteraturstudien (se Tabell 25). Relevansen hos de här och övriga kriterier som presenterats, analyseras i 8.1

Tabell 25. Kriterier identifierade via praktiken.

Kriterium	**Selenium**	**Ranorex**
1. Felbeteende kan anges för testfall i en testsvit, d.v.s. vad som ska hända när ett testfall misslyckas. [Testexekvering]	Nej.	Ja.
2. Antal iterationer för ett testfall kan konfigureras. [Testexekvering]	Nej.	Ja.
3. Tester för verktyget kan exekveras som en bakgrundsprocess. [Testexekvering]	Ej hittat information.	Ej hittat information.
4. Uppspelningshastigheten av testfall kan justeras för verktyget. [Testexekvering]	Ja.	Ja.
5. Regular Expressions kan användas i valideringsvillkor. [Testexekvering - Fånga och jämföra resultat]	Ja.	Ja
6. Screenshots kan visas i testrapporter för verktyget. [Testrapportering]	Nej.	Ja.
7. Verktyget kan visa grafisk testdata i testrapporter. [Testrapportering]	Nej.	Ja.
8. Verktyget kan automatiskt ta en bild över det som visades när ett fel uppstod. [Debuggingstöd]	Delvis.	Ja.

8 Resultat och slutsatser

I det här kapitlet diskuteras erhållna resultat, i vilka avseenden resultaten kan generaliseras till andra situationer, samt hur frågeställningen och de olika delmålen anses uppfyllda

8.1 Analys av identifierade kriteriers relevans

Ett viktigt begrepp inom forskning är *validitet*, som avser hur ackurat det som presenteras är i förhållande till verkligheten. I det här arbetet handlar validitet om huruvida de kriterier som presenteras faktiskt är relevanta för en utvärdering av GUI-testningsverktyg.

När det gäller selektionskriteriernas (se 6.1) relevans så har en motivering getts över att ISO/IEC 9126 är en mångsidig kvalitetsstandard som är relevant för all typ av programvara, vilket diskuteras av Bekhamal m.fl. (2005). De här kriteriernas relevans analyseras inte vidare då det redan finns mycket litteratur som diskuterar relevansen hos ISO/IEC 9126 och dess olika kriterier. De funktionella selektionskriterierna som är specifika för automatiserade GUI-testningsverktyg är dock värda att belysa. Det finns flera olika testningsaktiviteter som kan automatiseras, därför är det relevant att utvärdera vilka testaktiviteter ett verktyg förser stöd för (testexekvering, testdesign etc.). Det är även bra att utvärdera vilka typer av GUI-applikationer verktyget riktar sig mot, eftersom det skiljer sig åt. Tekniker som kan användas för att skapa testscript är relevant då det påverkar vilken kunskap användaren behöver. Gällande teknik för identifiering av GUI-komponenter så är det kritiskt att beakta då det påverkar hur testscripten kan hantera ändringar i GUI:t som berörs. Det finns fler kriterier som kan vara relevanta, men det här tillhör de som alltid är viktiga.

När det gäller relevansen hos de utvärderingskriterier som identifierats (se 6.2), så ges här en analys utifrån den abstrakta testprocess som illustreras och beskrivs i 2.2.4. Det här görs för att tydliggöra på vilket sätt kriterierna är relevanta i förhållande till den här testprocessen. Den här testprocessen innehåller steg som bör finnas i alla testprocesser. Hur kriterierna relaterar till en viss testprocess kan dock skilja sig åt i viss utsträckning.

8.1.1 Analys av utvärderingskriteriernas relevans

Med tanke på att testning upptar mycket av den totala tiden inom utvecklingsprojekt (Grindal m.fl., 2006), så är det viktigt att verktyget förser stöd för *testplanering och översikt* (se 6.2.1). Den organisatoriska testprocessen kan behöva justeras för ett specifik projekt exempelvis, därför är det bra om verktyget förser stöd för att göra det. När det gäller den applikationsdomän som avses så kan det ställa speciella krav på verktyget, vilket är värt att beakta, exempelvis för realtidssystem så blir tidsaspekter viktiga. För planeringen av testningen är det även viktigt att schemalägga testaktiviteter, att skapa en projektöversikt, och att hantera risker (Illes m.fl., 2005). Att verktyget kan uppskatta förväntad exekveringstid och mäta faktisk exekveringstid är även relevant då det påverkar hur mycket tid som behöver avsättas för att utföra testningen. Att verktyget förser stöd för att ange strukturen av testspecifikationer är relevant, då det är viktigt att det finns dokumenterat vad som ska testas och hur. Att dokumentation görs över det testningsarbete som ska utföras, är viktigt för att ge en översikt.

Analys och design (6.2.2) utgör en viktig del automationen av testning. Verktyg som automatiskt kan generera användbara testfall utifrån en modell kan vara tidsbesparande och kan även innebära att tester väljs på ett bättre sätt än om det görs manuellt (se 2.4.2). Ett

problem som Memon (2008) nämner är att det kan vara tidskrävande och svårt att skapa representativa modeller över GUI:n som har många möjliga tillstånd och tillståndsövergångar. Automatisk generering av en sådan modell genom analys av källkoden för GUI:t, kan därför vara värdefullt. Genering av högnivå testfall utifrån källkoden handlar också framförallt om att spara tid. Testfall kan genereras på flera sätt (se 2.2.1), det bör därför gå att ange för verktyget vilken testteknik som tester ska genereras utifrån, se exempelvis Hayat och Cadeer (2007). Vilka testfall som är relevanta beror på syftet. För belastningstester (load-testing) kan särskilda tester vara intressanta, t.ex. tester som motsvarar ett användarscenario som drar mycket serverresurser. Dessutom kräver sådan typ av testning att många användare kan simuleras med verktyget. Ytterligare är det värdefullt om verktyget kan avgöra vilka tester som är aktuella, samt att oanvändbara testfall kan repareras, när GUI:t har modifierats (Memon & Soffa, 2003; Memon, 2001). Det här kan spara tid i samband med regressionstestning.

Kriterier som är relevanta för *implementation av automationen* (se 6.2.3) omfattar bland annat tillvägagångssätt för att skapa testscript för verktyget. Att skapa automatiserade tester tar ofta mycket tid (Memon & Soffa, 2003). Det kan därför vara relevant att mäta hur lång tid det tar att automatisera ett antal tester (Börjesson & Feldt, 2012), dock bör man beakta att verktyg kan ha olika inlärningskurvor. För att snabbt kunna skapa testscript är det värdefullt om ett användarscenario kan spelas in (kräver inget kodningsarbete), eller om testscript kan genereras utifrån en modell. Det är även bra om det går att manuellt editera testscript med verktyget, då det ger en större kontrollbarhet och fler möjligheter. Keyword-driven testing förser en högre abstraktionsnivå över testfall genom att testfall representeras i form av nyckelord. Det här gör att personer som saknar programmeringskunskap kan skapa automatiserade tester, vilket kan vara önskvärt (se 7.1). När det gäller data-driven testing innebär det att ett testscript kan köras med olika testdata (input) från externa källor, vilket gör att flera likvärdiga script inte behöver skapas (Guo, Fu & Feng, 2010). Det här påverkar både implementationen, testexekveringen, och är även relevant ur underhållsperspektiv.

Testsviter bör kunna skapas och hanteras av verktyget, då ett antal tester ofta har ett gemensamt syfte, exempelvis att testa en viss komponent. Ytterligare är det relevant att utvärdera hur verktyget hanterar dynamiskt material vid konstruktion av testfall. Exempelvis om en inspelning av en användarsession görs, hur hanterar då verktyget material som måste laddas (Börjesson & Feldt, 2012). Hanteras inte tidsaspekter är det stor risk att återuppspelningen misslyckas – om verktyget utför saker vid fel tidpunkt etc. Det här är alltså relevant både vid implementation av testscripten men även för exekveringen av testscript. Bildigenkänning i IDE:n är relevant för verktyg som använder bildigenkänning för att identifiera GUI-komponenter vid testexekvering. Ändrar en GUI-komponent utseende bör det gå att omkoppla en händelse mot en ny bild genom verktygets IDE.

Att automatisera *testexekveringen* är viktigt då det är där den största möjliga återbetalningen med att automatisera finns (Wissink & Amaro, 2006). Relevanta kriterier att utvärdera kring testexekveringen (se 6.2.4) omfattar bland annat om det är möjligt att med verktyget exekvera testscript som tagits fram på olika sätt (inspelade, manuellt skapade etc.), då det påverkar vilka tillvägagångssätt som kan användas. Att det går att pausa och återuppta exekveringen är relevant om man vill övervaka exekveringen men behöver göra något annat en stund. Att övervaka testexekveringen kan framförallt vara intressant i samband med att testscripten konstrueras för att se att de fungerar (Bach, 1999). För att kunna övervaka testexekveringen är det även relevant att verktyget kan ge realtidsfeedback kring de

kommandon som körs etc. Tillståndet från vilken ett test utförs är kritiskt vid GUI-testning (Yuan m.fl., 2011). När en testsvit körs kan det därför vara önskvärt att verktyget kan försätta GUI-applikationen i dess ursprungstillstånd när ett fel inträffar för ett test (Hicks m.fl. 1997). Att testscript kan exekveras från en fjärrdator kan i många fall vara viktigt (Börjesson & Feldt, 2012). Exempelvis om ett verktyg endast finns installerat på en dator på ett företag, men det finns ett behov av att de kan exekveras av flera personer som eventuellt är lokaliserade på andra platser.

Fånga och jämföra resultat (se 6.2.5) är viktigt i samband med testexekvering, och vid analys av testningen. Att utvärdera tillståndet hos ett GUI vid exekvering, genom att jämföra tillståndet hos olika egenskaper med det förväntade tillståndet, är nödvändigt för att kunna uttala sig om GUI:t har förväntat beteende. För det här behövs det automatiserade orakel (se 2.2.3. Hur ett orakel utformas har stor inverkar på dess förmåga att upptäcka fel (Xie & Memon, 2007), det är därför bra om verktyget förser stöd för att skapa orakel. Det är även bra om verktyget samlar in loggningsinformation för testexekveringen, vilket kan vara värdefull information vid en analys etc.

Testrapportering (se 6.2.6) utgör en viktig del i testprocessen, där testrapporter ska tas fram för att det ska gå att ta del av vilken testning som utförts och dess utfall. Det är bra om verktyget kan skapa testrapporter automatiskt för att det ska gå att ta del av utförd testning, samt att säkerställa att det blir gjort. Persson och Yilmazturk (2004) nämner att en av de stora fördelarna med att använda verktyg för automatiserad testning är att testrapporter kan generas automatiskt. Går det att skapa testrapporter är det även relevant att utvärdera om det går att ange vad som ska sparas till testrapporterna, samt om det går ställa in vad som ska visas. Vilken information som är relevant kan skilja sig åt, beroende på vem som läser testrapporten etc. Att testrapporter automatiskt kan skickas till en angiven mailadress med hjälp av verktyge kan vara bra om testrapporterna behöver delas (Memon m.fl., 2003).

Utvärderingskriterier kring debuggingstöd och underhållsstöd (se 6.2.7 och 6.2.8) är inga testaktiviteter, men det är ändå viktigt att verktyget förser stöd för det. Testning följs vanligtvis av debugging, där koden för buggar ska spåras och åtgärdas. För *debuggingstöd* är det bra om felnoteringar kan genereras, och om buggar kan specificeras och prioriteras med hjälp av verktyg, för att få en översikt över vad som behöver göras. När det gäller underhållsstöd så är det viktigt av flera anledningar, exempelvis för att effektivisera testningsarbetet. Versionshantering är viktigt underhållsstöd för att kunna knyta tester mot en specifik version av en programvara, för att flera personer ska kunna ta del av dem, samt för att förse en spårbarhet över ändringar hos testobjekt (Guo, Fu & Feng, 2010). En testprocess består vanligtvis av en kedja av aktiviteter som hör ihop (se 2.2.4), det är därför bra om verktyget kan förse en spårbarhet mellan olika delar av testningen (ex. mellan krav, testfall). Snapshots-inrättningar innebär att en exakt kopia skapas för verktyget vid en viss tidpunkt (vilket omfattar inställningar och testscript verktyget använder etc.). Det kan vara relevant om en exakt kopia av testmiljön behövs, exempelvis om man skulle vilja laborera etc.

När det gäller de kriterier som identifierades via praktiken (se 7.3.1), så är de flesta kring testexekvering. Att felbeteende kan specificeras handlar om att det går det att ange vad som ska ske när ett testfall i en testsvit misslyckas (inget ska ske, resterande tester ska ej köras etc.). Det är relevant då det påverkar resterande tester i testsviten (Yuan m.fl, 2011). Antal iterationer som ett test ska köras kan vara relevant om man vill utföra något flera gånger. Regular Expressions (RE) är viktigt för den här typen av verktyg (Jia & Liu, 2002), eftersom

testoraklen (se 2.2.3) blir mer begränsad om inget stöd finns för att använda RE. Att grafisk testdata kan visas i testrapporter och att ett automatiskt screenshot kan tas i samband med fel, underlättar i debuggingarbete, vilket framhävs av Ruiz och Price (2008). Screenshot vid fel kan indikera på vilken typ av fel det handlar om. Exempelvis kan miljömässiga faktorer få ett test att misslyckas, till exempelvis att ett antivirus program körs igång automatiskt.

8.2 Analys av utvärderingen i fallstudien

I det här arbetet har en fallstudie utförts (se kapitel 7) där två Capture & Replay verktyg utvärderats. De selektionskriterier som användes i fallstudien sattes upp i samspråk med utvecklingschefen på det berörda företaget. Alla identifierade selektionskriterier har inte använts – vilka selektionskriterier som används vid en utvärdering bör anpassas efter de behov som finns (Poston & Sexton, 1992). Selektionskriterierna som användes i fallstudien kunde besvaras genom att läsa dokumentation, förutom kriteriet ”verktyget ska vara väldokumenterat” som måste bedömas. Kriterierna fungerade väl; en stor mängd av verktyg kunde sållas bort. Det fanns dock fler verktyg än de som utvärderades som uppfyllde uppsatta selektionskriterier, men det här var de som ansågs uppfylla dem bäst. Att göra en utvärdering för två verktyg ansågs tillräckligt för att visa på hur kriterierna kan användas.

När verktyg valts ut skulle de utvärderas. De kategorier av kriterier som är aktuella för de här verktygen är: testimplementering, testexekvering, fånga och jämföra resultat, testrapportering, debuggingstöd, samt underhållsstöd. De flesta av de identifierade kriterierna inom de här kategorierna har utvärderats. Många av kriterierna är av kvalitativ karaktär där ja/nej/delvis använts för att ange hur väl kriterierna uppfyllts. Delvis har indikerat på att kriteriet kan uppfyllas, i de flesta fall genom tillägg. Fördelen med att använda delvis vid bedömning är att det ger en mer rättvis bild av vissa verktyg. I den här fallstudien utvärderades ett open source-verktyg (Selenium) och ett kommersiellt (Ranorex). Det kommersiella verktyget uppfyllde betydligt fler kriterier, även om open source-verktyget kunde uppfylla de flesta kriterierna genom tillägg. Resultaten för de olika kriterierna kan jämföras på flera sätt (se 2.5). Här ges en kategorivis jämförelse.

När det gäller testimplementering så finns det en del skillnader. Ranorex har bra stöd för data-driven testing, data kan inhämtas från Excel-filer, CSV-filer och från SQL-databaser till de script som skapas. I Selenium kan data läsas in från XML-filer genom tillägg. Selenium använder inte bildigenkänning för identifiering så därför uppfylls inte det kriteriet. En annan stor skillnad är hur verktygen hanterar dynamiska objekt. I Ranorex går det utan problem, men i Selenium är det mer besvärligt. I Selenium behöver referenser till dynamiska objekt anges manuellt, dessutom behöver pauser läggas in för att dynamiskt innehåll ska hinna laddas. Det här gör det mer ansträngande och tidskrävande att ta fram testscript i Selenium.

Exekvering av testscript skiljer sig åt en del mellan verktygen. I Ranorex går det att ange vad som ska ske när ett testfall utförts (eller misslyckats) genom att skapa en teardown-region. Ytterligare går det att ange felbeteende för testfall, samt hur många gånger de ska köras. Ingen av de här möjligheterna finns i Selenium. När det gäller fånga och jämföra resultat (testexekvering) är den största skillnaden att i Ranorex ges stöd för att skapa orakel, genom att det med verktyget går att ange vad som ska valideras. Exempelvis för en textruta kan texten som ska valideras läsas in med verktyget. I Ranorex kan bildvalidering och egenskapsvalidering göras, medan i Selenium är det endast möjligt med egenskapsvalidering. Testrapportering är den kategori där det finns störst skillnader mellan verktygen. Selenium

har ingen funktionalitet för loggning eller generering av testrapporter per default, men det kan fås genom tillägg. Ranorex har god rapporteringsfunktionalitet. I testrapporterna visas vilka tester som körts, hur lång tid tog, vad som gått rätt och fel etc. I Ranorex kan även grafisk testdata visas i form av cirkeldiagram över utförda testfall och deras status.

Debuggingstöd är en kategori där verktygen presterar likvärdigt. Det stöd som de här verktygen ger är dels att felnoteringar kan genereras, samt att en bild kan tas automatisk när ett fel inträffar (tillägg krävs för Selenium). Slutligen när det gäller underhållsstöd, så förser båda visst stöd. Det finns dock bättre funktionalitet för det i Ranorex som har ett inbyggt gränssnitt mot SVN och TFS för versionshantering. Det format som scripten kan sparas till i Selenium ger dock möjligheter till att få versionshantering.

Sammantaget utifrån hur de båda verktygen fungerade med den webbapplikation som testades, så går det att konstatera att Ranorex presterar bättre. Den största fördelen med Ranorex är att dynamiskt material (Javascript etc.) kan hanteras väl. Det är mer problematiskt i Selenium, vilket gjorde att det tog förhållandevis lång tid att ta fram fungerande script. Andra fördelar med Ranorex är att god funktionalitet finns för testrapportering, och för data-driven testing.

8.3 Generalisering av resultaten

Inom forskning är det viktigt att kunna generalisera resultat; att resultaten kan appliceras i liknande situationer utanför studien. Från det här arbetet kan flera resultat generaliseras:

- Kriterier för utvärdering av GUI-testningsverktyg (selektionskriterier och utvärderingskriterier), samt hur de kan användas och kategoriseras.
- Den metodik som använts för att utvinna kriterier och utföra utvärderingen.

I det här arbetet har ett antal kriterier för utvärdering presenterats, både kriterier för att göra ett urval av verktyg (selektionskriterier) och kriterier för att utvärdera verktyg funktionellt (utvärderingskriterier). De här kriterierna kan återanvändas. Vilka kriterier som används och hur, bör dock anpassas efter situationen. En utvärdering bör därför inledas med någon sorts behovsanalys (Poston & Sexton, 1992). Intervjuguiderna som finns att ta del av i Appendix A och Appendix B kan återanvändas i samband med en behovsanalys etc. De kan dock behöva kompletteras med ytterligare frågor. De kriterier som är viktiga vid en utvärdering kan som konstaterat skilja sig åt, då organisationer har olika behov. Exempelvis tillgänglig budget för verktyg kan skilja sig åt, i vissa fall kanske endast open source-verktyg är aktuella. Det går dock inte att förutsäga resultatet av att använda de här kriterierna. Hur lyckad en utvärdering blir beror dels på vad som utvärderas, men även hur. De kriterier som presenterats i det här arbetet och hur de använts ska därför ses som ett stöd, snarare än ett facit.

Kriterierna från det här arbetet kan som sagt återanvändas för utvärderingingar, även om hur de används på ett lämpligt sätt kan tänkas skilja sig åt. Många av kriterierna är av karaktären ja/nej/delvis (grad av uppfyllelse). De kan återanvändas rakt av. Vissa kriterier så som ”applikationsspecifika karaktäristiker kan testas med verktyget” måste dock anpassas till situationen. Vad delvis innebär kan också variera. I det här fallet har det i de flesta fall inneburit att kriteriet kan uppfyllas med hjälp av tillägg (open source). Den tolkningen kan återanvändas. Kategoriseringen av kriterier är ett annat resultat som kan generaliseras. De kategoriseringar som använts för selektionskriterierna och utvärderingskriterierna är i sig

inget nytt. Det här arbetet ger dock en ökad förståelse kring hur olika aspekter hos GUI-testningsverktyg kan knytas till en testprocess och kvalitetskriterier enligt ISO/IEC 9126.

Den metodik som använts för att utvinna kriterier och i fallstudien kan även återanvändas. Beskrivet tillvägagångssätt för litteraturstudien kan även återanvändas. I Appendix A och Appendix B presenteras två intervjuguider som använts i det här arbetet. De frågorna kan återanvändas, även om det i vissa fall kan behövas ytterligare frågor. I det här fallet fanns lite förkunskaper om det företag, för vilket en utvärdering av verktyg gjordes (se kapitel Fallstudie). Intervjuerna syftade därför dels till att få kunskap om företagets testningsarbete och deras behov kring testautomation (för att kunna upprätta selektionskriterier etc.), samt till att kunna utvinna kriterier för utvärdering.

Sammanfattningsvis har det här arbetet bidragit med följande resultat:

- Selektionskriterier (se 6.1) indelade enligt ISO/IEC 9126, med utökade kategorier.
- Utvärderingskriterier (se 6.2) indelade enligt en abstrakt testprocess.
- En metodik som kan användas för att utföra en utvärdering, vilken omfattar:
    - Intervjufrågor (se Appendix A och Appendix B) som kan användas för att reda ut behov kring testautomatisering och verktyg för testautomation inom en organisation, samt för att kunna utvinna utvärderingskriterier.
    - Beskrivet tillvägagångssätt för litteraturstudien som kan återanvändas för at utvinna kriterier.
    - Förslag på hur delar av identifierade kriterier kan användas.
- Resultat från fallstudien (se 7.3 och Appendix C) som visar på hur delar av identifierade kriterier kan användas.

8.4 Analys av frågeställningen och delmål

Frågeställningen för det här arbetet är följande:

- Vilka kriterier är relevanta för en utvärdering av verktyg för automatiserad GUI-testning och hur kan dessa kriterier användas?

I det här arbetet har ett antal kriterier för utvärdering sammanställts, vilka har identifierats genom litteratur, intervjuer, och observationer. Kriteriernas relevans har analyserats utifrån litteraturen, och en fallstudie har utförts med syftet att undersöka och skildra om kriterierna kan användas och i så fall hur. Arbetets frågeställning har delats upp i delmål. Här ges en diskussion kring hur delmålen anses ha uppfyllts och därmed frågeställningen:

8.4.1 Delmål 1

Delmål ett lyder:

- Delmål 1 – Att identifiera och presentera selektions- och utvärderingskriterier för utvärdering av GUI-testningsverktyg.

Det här delmålet har uppfyllts genom en litteraturstudie (se 4.1), samt genom intervjuer och observationer i en fallstudie (se 4.2 och 7). Det gick att utvinna relativt många kriterier, särskilt från litteraturen. De kriterier som identifierats har sammanställs och för att göra det mer överskådligt och för att sätta kriterierna i ett sammanhang, så behövdes någon typ av kategorisering. Selektionskriterierna har delats in utifrån kvalitetsstandarden ISO/IEC 9126, medan utvärderingskriterierna delats in utifrån en abstrakt testprocess. Båda de här

indelningarna har motiverats och indelningarna belyser att det kan vara värt att beakta både programvarukvalitet, och testprocessen vid en utvärdering av testverktyg.

Delmålet anses uppfyllt genom att selektions- och utvärderingskriterier identifierats och presenterats på ett tydligt och motiverat sätt.

8.4.2 Delmål 2

Delmål två lyder:

- Att analysera relevansen hos de kriterier som identifierats.

Relevansen hos de kriterier som identifierats har analyserats utifrån den litteraturstudie som utförts. Ytterligare har en analys gjorts över den praktiska utvärdering som utfördes i fallstudien (se 8.2), som ytterligare belyser relevansen för vissa av kriterierna.

Den analys som ges i 8.1 fokuserar framförallt på relevansen hos de utvärderingskriterier som identifierats. Att selektionskriterierna inte analyseras mer utförligt motiveras av att det redan finns mycket litteratur som berör ISO/IEC 9126 och dess kriterier, då det är en mångsidig kvalitetsstandard som är aktuell för all typ av programvara. De funktionella selektionskriterierna analyseras emellertid, då de är specifika för den applikationsdomän som avses i det här arbetet och därför är viktigare att belysa. När det gäller utvärderingskriterierna som identifierats så analyseras deras relevans genom att en anknytning görs till den abstrakta testprocess som finns beskriven och illustrerad i 2.2.4. Den här analysen grundar sig i den litteraturstudie som utfördes. Den analys som ges är övergripande; varje kriteriums relevans återges inte specifikt utan en helhetsanalys görs per kategori av kriterier (för vardera tabell i 6.2, samt 7.3.1). I den här analysen återges dock relevansen hos merparten av de kriterier som presenterats. När det gäller de kriterier som identifierades praktiskt i fallstudien, så gjordes en utökad litteratursökning med syftet att få belägg för kriteriernas relevans. Det gick att hitta stöd för de flesta av kriterierna.

Delmålet anses uppfyllt genom att kriteriernas relevans analyserats, vilket återgetts i 8.1.

8.4.3 Delmål 3

Delmål tre lyder:

- Att utvärdera de relevanta kriterierna genom att visa på hur de kan användas i praktiken.

Det här delmålet har uppfyllts genom en fallstudie, där delar av de relevanta kriterierna (baserat på analysen i delmål 2) har utvärderats för två Capture & Replay verktyg. Utvärderingen begränsades till ett urval av för fallstudien relevanta kriterier. Det här gjordes för att återge hur det kan se ut vid en utvärdering – alla kriterier är inte relevanta vid alla utvärderingar. Det här beror på att företag har olika behov etc. Kriterierna har använts och bedömts enligt beskrivna sätt från litteraturen. Fallstudien (se 7) visade att åtminstone de kriterier som utvärderades, går att använda. En analys kring utvärdering ges i 8.2 för att tydliggöra hur väl kriterierna fungerat och hur kriterier kan skilja sig åt mellan olika verktyg.

Delmålet anses uppfyllt genom att en utvärdering har gjorts som visar på hur kriterierna kan användas. Dessutom ges en analys kring den här utvärderingen.

9 Diskussion

I det här kapitlet diskuteras arbetets utförande i stort. Samhälleliga, etiska, och vetenskapliga aspekter som beaktats i arbetsprocessen beskrivs, samt förslag på framtida arbete.

9.1 Arbetets utförande

Arbetet har mestadels gått bra; det har uppstått en del frågor och problem längs vägen, vilka har bemötts så väl det gått. Här följer lite reflektioner kring arbetets utförande.

När det gäller intervjuerna var det tänkt att de skulle utföras efter att litteraturstudien utförts. Anledningen till det var att de som intervjuades då skulle kunna få se de kriterier för utvärdering (utvärderings- och selektionskriterier) som identifierats, och ha något att relatera till. Dessutom skulle det bli enklare att utforma givande intervjufrågor. Av praktiska skäl passade det dock bättre att utföra intervjuerna vid ett tidigare skede. Därför utfördes litteraturstudien och intervjuerna parallellt med varandra. Andra kriterier som inte togs upp av litteraturstudien framkom ändå av intervjuerna. Frågorna som ställdes blev av det här skälet lite annorlunda jämfört med om litteraturstudien hade utförts i sin helhet innan.

Utförandet av intervjuerna var även bland det svåraste i det här arbetet. Dels att ta fram givande frågor, samt att veta vilken nivå intervjuerna skulle läggas på. Personen som ordnade med intervjupersoner gav en del information om vad de här personerna hade för ungefärlig kunskap inom området, vilket underlättade en del när intervjufrågorna skulle tas fram. En intervjuguide användes med frågor av kvalitativ karaktär. En kvalitativ ansats valdes då det möjliggör att frågor kan anpassas efter de svar som ges, och att svar kan motiveras. Intervjuer bör utformas på ett sätt som passar de som intervjuas. I det här fallet hade jag fått information om att de som intervjuades förväntades kunna en del kring testning och testautomation, men kanske inte specifikt om testverktyg. Det hade kanske varit mer givande att intervju personer med stor erfarenhet av testverktyg. Samtidigt återspeglade de här intervjuerna hur det kan se ut i verkligheten för företag. Intervjuerna bidrog framförallt till en ökad förståelse kring företagets testningsarbete och vilka behov de har kring testautomation. Intervjuerna visade sig dock vara begränsade i syftet att identifiera nya kriterier för utvärdering, även om de gav värdefull information för fallstudien.

När det gäller att utvinna kriterier så hade det även varit intressant att utföra en enkätundersökning, där personer hade fått ange hur relevanta de anser att identifierade kriterier är. Det hade kunnat ge en bild av vilka av kriterierna som anses särskilt viktiga. Gällande litteraturstudien så hade det varit intressant att göra en mer heltäckande studie inom området för att eventuellt identifiera fler relevanta kriterier (exempelvis söka i fler databaser). Det hade även varit intressant att gå igenom all den litteratur som refererar till den litteratur som används i det här arbetet, för att se om det skulle bidra med ytterligare kriterier. En del av dessa referenser har kollats upp, framförallt för den litteratur som många refererat till, vilket framförallt gjorts av nyfikenhet kring varför just den litteraturen är så populär. Att inte alla de referenserna kollats upp beror på att det inte ansågs viktigt när litteraturstudien utfördes. Vid mer eftertanke så skulle det dock varit bra.

När det gäller kategoriseringen av kriterier så har selektionskriterierna delats in enligt ISO/IEC 9126 som är en kvalitetsstandard för programvarukvalitet. Det finns väldigt många definitioner för vad kvalitet är och vad det inbegriper (Al-Kilidar, Cox & Kitchenham, 2005).

En brist i ISO/IEC 9126 som poängteras av Carvallo och Franch (2006) är att icke-tekniska kvalitetskriterier inte beaktas. Exempel på icke-tekniska kriterier är sådana som berör leverantören, exempelvis gällande vilken support som erbjuds för produkten, vilket rykte leverantören har etc. Varför kvalitet definieras på olika sätt kan bland annat bero på att olika perspektiv används. McCalls modell har ett produktperspektiv, som exempelvis omfattar huruvida en programvara är korrekt med avseende på produktspecifikationen (Bekhamal m.fl., 2005). ISO/IEC 9126 har istället ett användarperspektiv, där endast kriterier som en användare kan utvärdera på egen hand finns med.

Att just ISO/IEC 9126 använts i det här arbetet motiveras av att den representerar ett användarperspektiv och att det är en väletablerad standard. Den är dock inte komplett, vilket blir tydligt av att en utökning behövde göras med icke-tekniska kvalitetskriterier kring pris och licensiering, samt leverantören, som utgör viktiga faktorer vid en verktygsutvärdering. När det gäller utvärderingskriterierna som identifierats så har utgångspunkten (en abstrakt testprocess) påverkat vilka kriterier som varit relevanta. I det här fallet var en indelning enligt en testprocess lämplig då de verktyg som avses är testverktyg. Det är viktigt att poängtera att utöver de kriterier som presenteras i det här arbetet så kan det finnas andra kriterier som kan vara relevanta beroende på vilken utgångspunkt som används.

Fokus för arbetet har skiftat under arbetets gång. Först var det tänkt att en utvärdering skulle göras för ett par Capture & Replay verktyg. I litteraturen visade det sig dock finnas en oenighet kring vad som är värt att utvärdera, och hur. Litteraturen tar dessutom upp att val av verktyg för testautomatisering är kritiskt, och att ”fel” verktyg införskaffats kan vara en av huvudanledningarna till en bristfällig testautomatisering. En utvärdering är i de fallen värdefullt, och utvärderingens utförande påverkar i stor utsträckning huruvida ”rätt” verktyg identifieras. Arbetets fokus ändrades därför till själva utvärderingsprocessen istället.

9.2 Samhälleliga, etiska och vetenskapliga aspekter

Under arbetets gång har samhälleliga, etiska och vetenskapliga aspekter beaktats. När ett arbete av den här typen utförs är det viktigt att det har någon samhällelig nytta, samt att etiska och vetenskapliga aspekter respekteras. Vilken nytta ett arbete har är kanske det första man bör fundera över. I det här fallet kan det utvärderingsverktyg som tagits fram användas som stöd vid en utvärdering av automatiserade testverktyg. Dels kan det underlätta för den som använder utvärderingsverktyget att veta vad som ska kan utvärderas, och även hur. Dessutom kan den metodik som använts i det här arbetet återanvändas. Det utvärderingsverktyg som presenterats i det här arbetet kan i bästa fall innebära att den som använder det identifierar ett verktyg som uppfyller den behovsbild som finns.

Arbetet har utförts metodiskt och vetenskapligt. Problemet grundar sig i att det i litteraturen nämns att en risk vid testautomatisering är att fel verktyg införskaffats pågrund av bristfälliga eller uteblivande utvärderingar. Det här arbetet syftar därför framförallt till att bidra till en ökad kunskap kring vad som kan vara värt att utvärdera för ett sådant verktyg. Problemet har angripits på två sätt: dels teoretiskt genom litteraturen, samt genom ett praktiskt perspektiv i en fallstudie. Anledningen till varför båda de här perspektiven använts är för att det kan finnas en skillnad mellan vad som förespråkas i praktiken och det som förespråkas i litteraturen. Problemställningen ansågs besvarad när de delmål som satts upp blivit uppfyllda. Anledningen till varför delmål sattes upp var dels för att underlätta i arbetet, men även för att göra framställningen tydligare.

Etiska aspekter beaktades särskilt vid intervjuerna. De som intervjuades informerades om syftet med intervjun, om hur personliga uppgifter kommer hanteras, samt att de under intervjun har rätt att när som helst avsluta intervjun eller avstå att svara på frågor. De har även fått reda på hur intervjun skulle dokumenteras. Intervjuerna dokumenterades genom anteckningar. Ingen inspelning gjordes, eftersom det dels skulle kunna göra intervjupersonerna mindre avslappnade, samt eftersom det skulle det ta lång tid att transkribera intervjuerna. Det ansågs inte heller att inspelning skulle bidra något nämnvärt jämfört med om endast anteckningar fördes. Båda teknikerna har för- och nackdelar. Sker ingen inspelning innebär det att man måste lyssna mer aktivt och verkligen vara koncentrerad. Nackdelen i de fallen är att det kan vara svårt att hänga med i allt som sägs, då anteckningar ska göras samtidigt. Inspelning av intervjuer har den stora fördelen att det går att lyssna på intervjun i efterhand om något skulle ha missats.

Arbetet utfördes delvis på ett företag där personer intervjuades och där verktyg utvärderades mot en av företagets applikationer. Integriteten för företaget och de personer som berörts av det här arbetet har respekterats, till exempel nämns inte personer och företag vid namn i den här rapporten. Det har gjorts för att det inte ska gå att veta vilket företag och vilka personer som avses, vilket ändå är irrelevant för arbetet. Den applikation som testats framgår inte heller av det här skälet. Personerna som berörts av det här arbetet har informerats om att de uppgifter som berör dem och företaget ska hållas konfidentiellt. Genom att värna om personer och företags integritet kan personer vara mer samarbetsvilliga.

9.3 Framtida arbete

Det finns olika typer av framtida arbeten som skulle kunna utföras utifrån det här. Dels skulle resultaten kunna valideras ytterligare, framförallt de utvärderingskriterier som inte utvärderats i det här arbetet. Det här skulle förslagsvis kunna göras för en annan typ av verktyg, till exempel ett verktyg som kan generera testfall. Ett annat alternativ vore att testa de här kriterierna för en särskild applikationsdomän, till exempel för mobilapplikationer. De utvärderingskriterier som tagits upp i det här arbetet kan även användas vid utformandet av ett testautomatiseringsramverk (se 2.4.3). De här utvärderingskriterierna kan då fungera som en slags riktlinjer i arbetet och när ramverket är färdigbyggt kan en diskussion föras kring hur väl dessa kriterier fungerar i det avseendet.

En annan typ av arbete är att se vilka ytterligare kriterier som går att identifiera för GUI-testningsverktyg. Det utvärderingsverktyg som presenteras här är som nämnt utbyggbart, och det finns mer litteratur att studera etc. Det här utvärderingsverktyget riktar sig framförallt mot GUI-testningsverktyg (även om många kriterier är aktuella för andra typer av testverktyg också). Liknande utvärderingsverktyg som riktar sig mot andra typer av testverktyg skulle kunna vara till nytta. Sådana verktyg kan tas fram enligt samma metodik som använts i det här arbetet, men det skulle även kunna göras på ett annat sätt, till exempel genom enkätundersökningar. En kartläggning över GUI-testningsverktyg vore även intressant, genom att använda de kriterier som nämns i det här arbetet.

I det här arbetet är merparten av de kriterier som presenterats av kvalitativ karaktär. Ett utvärderingsverktyg som endast består av kvantitativa kriterier skulle även vara intressant. Ett sådant verktyg skulle kunna ge mer exakta svar om vilket verktyg som presterar bäst. Dock går det inte att komma ifrån att många av de kriterier som är viktiga vid en utvärdering av testverktyg är av kvalitativ karaktär.

Referenser

Ackerman, A. F, Buchwald, L. S. & Lewski, F. H. (1989) Software inspections: an effective verification process. *IEEE Software* 6 (3), pp. 31-36.

Al-Kilidar, H., Cox, K., & Kitchenham, B. (2005). The use and usefulness of the ISO/IEC 9126 quality standard. *In 2005 International Symposium on Empirical Software Engineering*, pp 126-132.

Ammann, P & Offutt, J. (2008) *Introduction to software testing*. Cambridge: Cambridge University Press.

Arlt, S., Bertolini, C. & Schäf, M. (2011) Behind the scenes: An approach to incorporate context in GUI test case generation. *Proceedings - 4th IEEE International Conference on Software Testing, Verification, and Validation Workshops, ICSTW*, 2011, art. nr. 5954413, pp. 222-231.

Bach, J. (1999) Test Automation Snake Oil, *14th International Conference and Exposition on Testing Computer Software*, Washington, DC, 1999.

Behkamal, B., Kahani, M. & Akbari, M.K. (2009) Customizing ISO 9126 quality model for evaluation of B2B applications. *Information and Software* Technology, 51 (3), 2009, pp. 599-609.

Beizer, B. (1990) *Software Testing Techniques* (2:a upplagan). New York: Van Nostrand Reinhold.

Berezin, S. (2002) *Model checking and theorem proving: a unified framework*. PhD thesis, Carnegie Mellon University.

Brooks, P.A. & Memon, A.M. (2007) Automated GUI testing guided by usage profiles *ASE'07 - 2007 ACM/IEEE International Conference on Automated Software Engineering*, pp. 333-342

Börjesson, E. & Feldt, R. (2012) Automated system testing using visual GUI testing tools: A comparative study in industry. *Proceedings - IEEE 5th International Conference on Software Testing, Verification and Validation, ICST*, 2012 , art. nr. 6200127 , pp. 350-359 .

Carvallo, J. P. & Franch, X. (2006) Extending the ISO/IEC 9126-1 quality model with non-technical factors for COTS components selection. *Proceedings of the 14th international workshop on Software quality, RE,* 2006, pp 9-14.

Claessen, K. & Hughes, J. 2011.QuickCheck: A lightweight tool for random testing of Haskell programs. *ACM SIGPLAN Notices* 46 (4), pp. 53-64.

Grechanik, M., Xie, Q. & Fu, C. (2009) Maintaining and evolving GUI-directed test scripts. *Proceedings - International Conference on Software Engineering, ICSE*, 2011, art. nr. 5070540, pp. 408-418.

Grindal, M., Offutt, J. & Mellin, J. (2006) On the testing maturity of software producing organizations. *Proceedings - Testing: Academic and Industrial Conference - Practice and Research Techniques, TAIC PART*, 2006, art. nr. 1691684, pp. 171-180.

Guo, W., Fu, X. & Feng, J. (2010) A data-driven software testing tools integration system. *International Conference on Computational Intelligence and Software Engineering, CiSE* 2010 , art. no. 5676878

Hayat, M.U. & Qadeer, N. (2007) Intra Component GUI Test Case Generation Technique. *2007 International Conference on Information and Emerging Technologies, ICIET*, 2007, art. nr. 4381328, pp. 153-156.

Hass, A. M. J. (2008). *Guide to Advanced Software Testing*. Norwood: Artech House.

Hicks, I. D., South, G. J. & Oshisanwo, A. O. (1997) Automated testing as an aid to systems integration. *BT Technology Journal*, *15* (3), 1997 ,pp. 26-36.

Heiskanen, H., Jääskeläinen, A. & Katara, M. (2010) Debug Support for Model-Based GUI Testing. *ICST 2010 - 3rd International Conference on Software Testing, Verification and Validation*, 2010, art. nr. 5477102 , pp. 25-34.

Heitlager, I. Kuipers, T. & Visser, J. (2007) A Practical Model for Measuring Maintainability. *QUATIC 2007 - 6th International Conference on the Quality of Information and Communications Technology,* 2007, art. nr. 4335232 , pp. 30-39.

Illes, T., Herrmann, A., Paech, B. & Ruckert, J. (2005) Criteria for software testing tool evaluation - A task oriented view. *Proceedings of the 3rd World Congress for Software Quality*, volym 2, 2005, 26-30 september, pp. 213–222.

Jia, X. & Liu, H. (2002) Rigorous and Automatic Testing of Web Applications. *In 6th IASTED International Conference on Software Engineering and Applications*, pp 280–285,

Lewis, W. E. (2009) *Software Testing and Continuous Quality Improvement* (3:e upplagan). Boca Raton: Taylor & Francis Group.

Memon, A. M. (2001) *A Comprehensive Framework For Testing Graphical User Interfaces.* PhD thesis, University of Pittsburgh.

Memon, A. M. (2002) GUI testing: pitfalls and process. *Computer*, 37 (8), 2002, pp. 87-88.

Memon, A. M., Banerjee, I. & Nagarajan, A. (2003) GUI Ripping: Reverse Engineering of Graphical User Interfaces for Testing. *Reverse Engineering - Working Conference Proceedings*, 2003, pp. 260-269.

Memon, A. M. & Soffa, M.L. (2003) Regression testing of GUIs. *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pp. 118-127.

Memon, A. M., Pollack, M. E & Soffa, M.L. (2001) Hierarchical GUI Test Case Generation Using Automated Planning. *IEEE Transactions on Software Engineering 27* (2), pp. 144-155.

Memon, A.M. (2008) Automatically repairing event sequence-based GUI test suites for regression testing. *ACM Transactions on Software Engineering and Methodology* 18 (2), art. nr. 4.

Namachivayam, N. (2012) Test Automation Life Cycle. *QAInsights* [Blogg]. 3 november. Tillgänglig på Internet: http://qainsights.com/test-automation-life-cycle/ [Hämtad 13.06.04].

Persson, C. & Yilmazturk, N. (2004) Establishment of automated regression testing at ABB: industrial experience report on 'avoiding the pitfalls'. *International Conference on Automated Software Engineering*, 20-24 september, pp. 112-121.

Poston, R. M. & Sexton, M. P. (1992) Evaluating and selecting testing tools. *IEEE Software*, *9* (3), pp. 33-42.

Pressman, R. (2010) *Software Engineering: A Practitioner's Approach* (7:e upplagan). New York: McGraw-Hill.

Ramler, R. & Wolfmaier, K. (2006) Economic Perspectives in Test Automation: Balancing Automated and Manual Testing with Opportunity Cost. *Proceedings - International Conference on Software Engineering*, 2006, pp. 85-91.

Ruiz, A. & Price, Y.W. (2008) GUI testing made easy. *Practice and Research Techniques, TAIC PART '08. Testing: Academic & Industrial Conference*, pp 99-103, Windsor, UK., August, 2008.

Wissink, T. & Amaro, C. (2006) Successful test automation for software maintenance. *IEEE International Conference on Software Maintenance*, ICSM, 2006, art. nr. 4021345 , pp. 265-266.

Xie, Q. & Memon, A.M. (2007) Designing and comparing automated test oracles for GUI-based software applications. *ACM Transactions on Software Engineering and Methodology*, *16* (1), art. nr. 4.

Yuan, X., Cohen, M.B. & Memon, A.M. (2011) GUI Interaction Testing: Incorporating Event Context**.** *IEEE Transactions on Software Engineering*, *37* (4), 2011, art. nr. 5444885, pp. 559-574.

Appendix A - Intervjuguide, behovsanalys

Innan tester automatiseras och ett verktyg för testautomation införskaffas, är det viktigt att göra någon sorts behovs- och nulägesanalys. En sådan analys kan öka chanserna till en lyckosam automatisering och bidra till en ökad kvalitet hos testningen. I det här arbetet utfördes en fallstudie, med det huvudsakliga syftet att se huruvida de kriterier som identifierats kan användas, och ge användbar information vid en utvärdering. Det är viktigt att känna till kontexten för en utvärdering. En intervju utfördes därför, för att få bättre kunskap om företagets nuvarande situation, och deras behovsbild. Här följer den intervjuguide som användes. För varje fråga ges en kommentar till varför frågan kan vara värd att ställa. De här frågorna kan så klart behöva kompletteras med ytterligare frågor. Vilka frågor som bör ställas beror på vilka kunskaper man har och vad man vill få ut av intervjun.

- **Varför automatisera?**

*Det kan finnas många anledningar till varför en organisation vill automatisera, till exempelvis med syftet att minska testningskostnaderna eller för att man vill kunna utföra mer testning etc. Det är viktigt att syftet klargörs för att veta vad som behöver uppfyllas.

- **Har ni någon automatiserad testning i nuläget? Om ja, vad för typ av automation och hur anser du att den fungerar?**

*Har organisationen i nuläget någon automatiserad testning är det värt att analysera hur den fungerar.

- **Vad för typ av testning anser du är bra att automatisera?**

*Det finns många typer av testautomation, och i de flesta fall många möjliga tester att automatisera. Det är viktigt att ”rätt” tester automatiseras för att automationen ska vara lönsam. Genom att ställa den här frågan kan en kunskap fås om vilka typer av tester som avses automatiseras (regressionstester etc.).

- **Vad finns det för risker med att automatisera?**

*Det finns många risker förknippade med att automatisera testning. Genom att ställa den här frågan kan man få en kännedom om huruvida den som intervjuas är medveten om att testautomatisering inte är riskfritt.

- **Vilka svårigheter finns det med att genomföra testautomatisering?**

*Den här frågan kan belysa svårigheter med att automatisera, och kan eventuellt ge värdefull information kring svårigheter organisationen upplevt kring att automatisera.

- **Vilka kunskaper måste den som utför automatiseringen besitta? (programmeringskunskap etc.)**

*Den här frågans syfte är att ge kunskap om vad den som utför automatiseringen förväntas ha för kunskap. De verktyg som finns för testautomation ställer olika krav på användarna. Den här frågan kan därför ge information som är viktig när verktyg väljs ut.

- **Vad är viktigast hos ett verktyg för automatiserad GUI-testning anser du? Att det kan användas för att testa många typer av GUI-applikationer, eller att det endast kan användas för en typ av GUI-applikation (t.ex. webbapplikationer) men att verktyget erbjuder mer funktionalitet anpassat för den typen av applikation. Motivera gärna ditt svar.**

*Det finns många verktyg för automatiserad GUI-testning. Vissa riktar sig mot en speciell typ av GUI-applikation, t.ex. webbapplikationer, medan andra är mer generella och kan användas för flera typer av GUI-applikationer. Den här frågan kan ge information om vilka typer av applikationer ett eventuellt verktyg måste hantera.

- **Vad är viktiga egenskaper hos ett ramverk för testautomatisering?**

*Är det tänkt att man ska använda ett ramverk för testautomatisering är det viktigt att känna till vad det bör innehålla, d.v.s. viktiga egenskaper.

- **Vilka verktyg använder ni huvudsakligen i er utvecklingsprocess?**

*Den här frågans syfte är att ge kunskap om vilka verktyg som redan används i utvecklingen. Det är ofta önskvärt att olika verktyg kan samverka, och därför krävs det att man reder ut vilka integreringsmöjligheter som måste finnas för ett nytt verktyg. Dessutom kan det vara så att man redan har licenser för ett visst verktyg för testautomatisering, och då är det dumt att införskaffa nytt utan att ha beaktat det befintliga.

- **Hur gör man lämpligen för att hitta ett verktyg som passar in i nuvarande utvecklingsprocess?**

*Den här frågan kan ge kunskap om hur man kan identifiera ett verktyg som passar in i utvecklingsprocessen.

Appendix B - Intervjuguide, identifiering av kriterier för utvärdering.

Ett par intervjuer utfördes med det huvudsakliga syftet att kunna identifiera kriterier som kan användas vid utvärdering av GUI-testningsverktyg. Dessutom skulle de här intervjuerna ge ytterligare kunskap om företaget för fallstudiens (se kapitel 7) testningsarbete och deras behov av testautomation och ett verktyg för att kunna automatisera. Skillnaden mellan de här intervjufrågorna och de i Appendix A är att de syftar till att ge en helhetsbild av företagets testningsarbete och behovsbild. De här frågorna å andra sidan syftar till att kunna utvinna kriterier för utvärdering, samt att ge mer detaljkunskap om företagets testningsarbete. Intervjuerna har alltså liknande syfte och därför påminner frågorna en del om varandra.

En intervjuguide användes, med frågor indelade i tre kategorier: testning generellt, automatiserad testning, samt testverktyg. Vad man bör tänka på är att innan frågor ställs om automatiserad testning och testverktyg kan det vara bra att börja generellt kring testning. Det här för att få en ökad förståelse kring organisationens testningsarbete, och för att se hur väl insatt den som intervjuas är i området. I den fallstudie som utfördes var det mest lämpligt med frågor om testning generellt, då det var oklart hur mycket de som intervjuades kan om testautomatisering och testverktyg. Vilka typer av frågor som bör ställas måste anpassas efter kunskapen hos de som intervjuas. Vet man att den som intervjuas kan mycket om testautomation och testverktyg, kan det vara värt att fokusera mer på de områdena, än testning generellt. Nedan motiveras varför de olika frågorna kan vara värda att ställa.

Testning generellt

- **Vilken arbetsroll har du, och på vilket sätt påverkar testning dig i ditt arbete?**

*Den här frågan ställs för att få kunskap om vilken arbetsroll en person har, och hur testning påverkar personen i sitt arbete.

- **Vad innebär testning för dig?**

* Det finns olika sätt att se på testning. Syftet med den här frågan är att få kunskap om en persons uppfattning av testning.

- **Utför du någon testning i ditt vardagliga arbete? Om ja - Hur mycket tid lägger du uppskattningsvis per vecka på testning?**

* Den här frågan ställs för att få mer kunskap om en persons relation till testning i sitt arbete.

- **Hur jobbar ni med testning på företaget i stort? Det vill säga vilka utför testning och vem har ansvaret för att det blir gjort?**

* Den här frågan ställs för att få en övergripande koll på hur ett företag jobbar med testning.

- **Hur väljs vad som ska testas?**

*Den här frågan ställs för att få kunskap om hur valet av vad som ska testas går till. För testautomation är det viktigt att det finns en tydlig plan kring vad som ska testas.

- **Finns det dokumenterat vad som behöver testas, och vilket resultat som förväntas få av ett testfall?**

* Hur testning dokumenteras påverkar starkt dess kvalitet, det är även viktigt om tester ska kunna automatiseras. Det är även viktigt att det finns specificerat vilka utfall som förväntas av olika testfall. Den här frågan kan ge svar på det.

- **Utförs testning utöver vad som kan anses vara det förväntade användarbeteendet vid testning av ett grafiskt användargränssnitt?**

*Grafiska användargränssnitt kan vanligtvis användas på många sätt, d.v.s. det finns många inputmöjligheter. Den här frågan syftar därför till att ta reda på huruvida mer testning sker, utöver vad som kan anses representera standardanvändningen.

- **Dokumenteras den testning som utförs och resultatet av den?**

*Det är viktigt att testning dokumenteras så man vet vad som har testats, och dess resultat. De rutiner som finns kring dokumentationen kan påverka de krav som ställs på ett verktyg för automatiserad GUI-testning, där testrapporter är viktiga.

- **Vem förväntas fixa problemet som upptäcks av testning?**

*Syftar till att ge kunskap om vem som förväntas fixa fel som upptäcks av testning. Särkskilda krav på felrapporteringen hos ett verktyg kan ställas beroende på vem som förväntas fixa felen.

- **Hur mycket av den testning som utförs är återkommande, och hur mycket är specifik för ett visst fall (t.ex. kopplat till ett projekt).**

*Den här frågan syftar till att få kunskap om huruvida det finns behov av testautomation.

- **Anser du att det finns något i företagets testningsarbete som skulle kunna förbättras, i så fall vad?**

*Den här frågan syftar till att få information om vad företaget kan förbättra i sitt testningsarbete.

Automatiserad testning

- **Är automatiserad testning något ni funderat över, och är det något som används i nuläget i något avseende?**

*Den här frågan ställs för att få kunskap om huruvida testautomation är något som beaktats, och om det används i nuläget i någon form.

- **Hur fungerar den testautomation ni har, och är det något du anser vore bra att ha i större utsträckning?**

*Har företaget i nuläget någon form av testautomation, kan vidare frågor ställas kring det. Till exempel den här.

- **Vilka fördelar ser du med automatiserad testning?**

*Det kan finnas många fördelar med att ha automatiserad testning. Det är bra om det finns en medvetenhet kring vad testautomatisering kan bidra till. Syftet med den här frågan är att ge kunskap kring de fördelar intervjupersonen ser med testautomation.

- **Vad finns det för risker med att automatisera?**

*Det finns många risker förknippade med att automatisera testning. Genom att ställa den här frågan kan man få en kännedom om den som intervjuas är medveten om att testautomatisering inte är riskfritt.

- **Är det uppenbart för vilka delar av era programvaror som testautomation skulle vara lämpligt?**

*Det är viktigt att ”rätt” tester automatiseras. Den här frågan syftar till ge information om hur uppenbart det är kring vilka tester som bör automatiseras.

Testverktyg

- **Har du erfarenhet av något/några testverktyg? Om ja – vilka är dina upplevelser kring det/de?**

*Den här frågan syftar till att ta del av intervjupersonens erfarenheter av testverktyg (om några).

- **Testverktyg kan användas för testning av grafiska användargränssnitt, är det ett användningsområde du har något erfarenhet av eller funderat över?**

*Det finns många typer av testverktyg. Den här frågans syfte är reda ut om intervjupersonen funderat över GUI-testning som ett användningsområde för testverktyg.

- **Vilka egenskaper anser du är viktiga hos testverktyg i allmänhet, och finns det några speciella egenskaper som är viktiga hos ett verktyg för automatisered GUI-testning?**

*Den här frågans syfte är att ge kunskap om vad intervjupersonen anser är viktigt hos testverktyg, med visst fokus på verktyg för GUI-testning.

Appendix C - Utvärdering av utvärderingskriterier, beslutsunderlag.

I den fallstudie som utfördes (se kapitel 6) utvärderades ett urval av kriterier för två Capture & Replay verktyg. Det här urvalet baserades på vad som var relevant i fallstudien, där en utvärdering gjordes med syftet att identifiera ett verktyg för automatiserad GUI-testning åt ett företag. Den här utvärderingen innebar att ett antal kriterier utvärderades praktiskt för två verktyg för att se om de här kriterierna är relevanta, och går att använda vid en utvärdering. Vilka kriterier som är relevanta beror så klart på situationen, samt beroende på vilken typ av verktyg som avses (alla kriterier applicerar inte på alla verktyg). Den här utväderingen finns att ta del av i 7.3, där de olika kriterierna besvaras kortfattat. Här presenteras tabeller med mer utförliga svar.

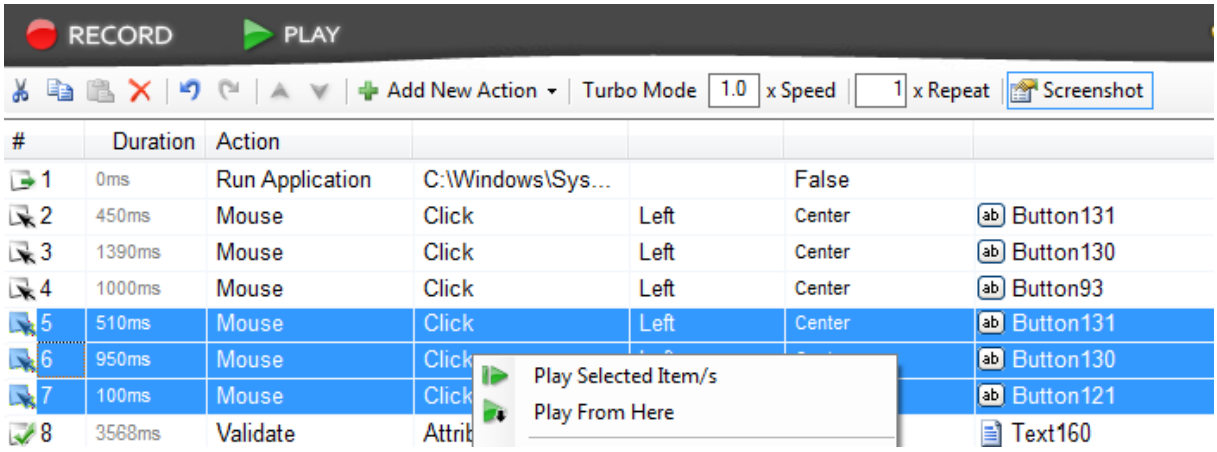
Tabell 26. Implementation av automatiseringen - beslutsunderlag.

Kriterium	**Selenium**	**Ranorex**
1. Verktyget möjliggör inspelning av exekverbara testfall.	Ja, inspelning kan göras av webbtester, dock endast i Firefox.	Ja, inspelning kan göras både för webbapplikationer, skrivbordsapplikationer, och mobilapplikationer.
2. Testscript går att editera med verktyget.	Testscripten går att editera i verktyget genom att olika parametrar ändras (keyword-driven testning används). I Figur 7 visas Selenium IDE. Direkt editering av källkoden går inte att göra i verktyget, men kod kan genereras och sedan editeras med hjälp av ett annat verktyg, till exempel Visual Studio.	Testscript går att editera på flera sätt: dels kan parametrar ändras (keyword-driven testing används), och dels kan direkta ändringar göras i källkoden.
3. Konkreta testfall kan genereras av verktyget utifrån modeller.	Nej.	Nej.
4. Verktyget har stöd för keyword-driven testing.	Ja, verktyget kommer med ett gränssnitt för att utföra keyword-driven testing.	Ja, verktyget kommer med ett gränssnitt för att utföra keyword-driven testing.
5. Verktyget har stöd för data-driven testing.	Det finns ingen inbyggd funktionalitet för data-driven testing, m.h.a olika tillägg kan det dock fås, se till exempel: http://unmesh.me/2012/12/04/data-driven-testing-with-selenium-ide/	Ja. Det finns möjligheter till data-driven testing genom att data läses in från CSV-filer, Excel-filer, eller från SQL-databaser.
6. Antal rader kod för genererade script (fem användarinteraktioner spelades in).	241	505

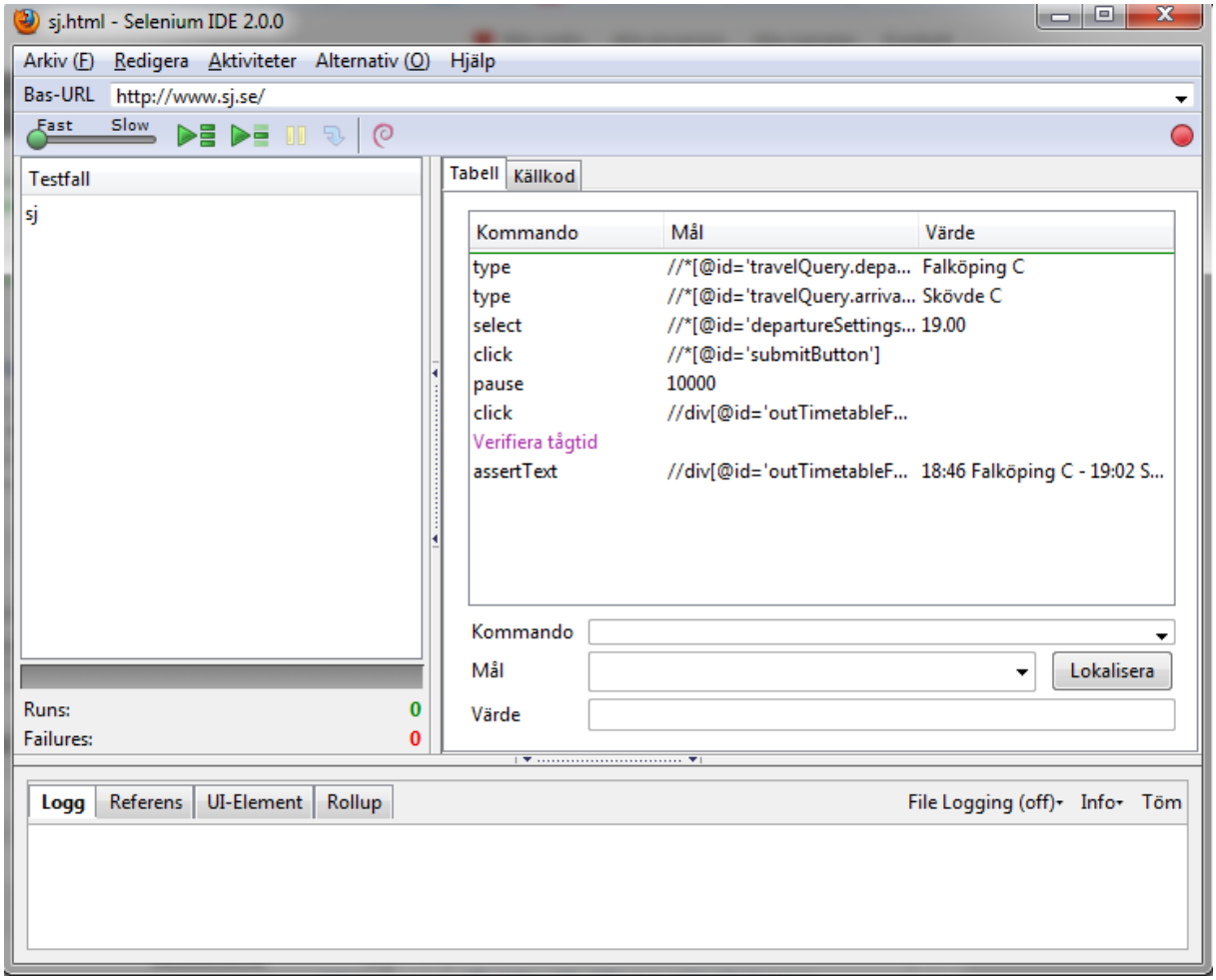
7. Verktyget har bildigenkänning i IDE	Nej, Selenium använder inte bildigenkänning som identifieringsteknik.	Ja, bilder lagras i en så kallad ”reposityory” där det bland annat går att uppdatera mappningen mellan kommandon och bilder.
8. Verktyget möjliggör inspelning av tester på en fjärrdator.	Ja.	Ja.
9. Verktyget har testsvit support.	Ja, det går att skapa testsviter som innehåller ett antal testfall. Varje testfall består alltid av ett användarscenario.	Ja, det finns support för att skapa och hantera testsviter. En testsvit består av testfall som i sin tur kan bestå av en eller flera användarscenarion.
10. Dynamiska GUI-komponenter kan hanteras av verktyget (t.ex. drop-down menyer).	Går med vissa besvär. Pauser måste läggas in eftersom verktyget utför olika kommandon så snabbt som möjligt. Det kan leda till problem för objekt som laddas dynamiskt, till exempel på en webbsida.	Går utan problem. Vid återuppspelning kommer händelser utföras under samma tid som vid inspelning. Hittas inte ett objekt kommer nya försöka göras under 30 sekunder, innan testfallet anses misslyckat. Det går att ställa in tiden mellan händelser, samt hur länge objekt ska eftersökas.

I Tabell 26 presenteras beslutsunderlag för kriterier som berör implementation av testautomatiseringen. Det finns en del nämnvärda skillnader mellan verktygen. En stor skillnad är att Selenium endast riktar sig mot webbapplikationer, medan Ranorex kan användas för att testa flera typer av GUI-applikationer. Det finns även skillnader i hur testscript kan ändras. I Selenium kan testscript ändras genom att olika parametrar modifieras (se Figur 7). Genom att generera kod kan även ändringar göras direkt i källkoden, till exempel i Visual Studio. Det här kräver dock att extra programvara installeras och konfigureras, se http://docs.seleniumhq.org/projects/webdriver/. Ranorex kommer med en IDE, och ändringar för testscript kan göras dels för olika parametrar (se Figur 6), men även direkt för källkoden. När det gäller antalet rader för genererade script så fanns stor skillnad. Tre likadana testfall skapades för Selenium och Ranorex. Scripten i Selenium innehöll 241 rader kod, medan motsvarande siffra var 505 för Ranorex.

När det gäller data-driven testing förser Ranorex bra stöd för det. Data kan läsas in från CSV-filer, Excel-filer, och från SQL-databaser. Selenium har visst stöd för data-driven testing via tillägg, men endast för att läsa från XML-filer. Andra skillnader är att Ranorex använder bildigenkänning för att identifiera GUI-komponenter, medan Selenium endast använder HTML-referenser. En betydande skillnad finns också kring hur verktygen hanterar dynamiska objekt. I Selenium måste pauser läggas in manuellt för att dynamiskt innehåll ska hinna laddas, medan i Ranorex kan det hanteras automatiskt av verktyget.



Figur 6. Ranorex gränssnitt för keyword-driven testing.



Figur 7. Selenium IDE med ett script för att verifiera en tågtid.

Tabell 27. Testexekvering - beslutsunderlag.

Kriterium	Selenium	Ranorex
1. Exekvering av testscript för verktyget kan ske via en fjärrdator.	Nej, inte för Selenium IDE. Det går dock för Selenium RC (http://docs.seleniumhq.org/docs/05_selenium_rc.jsp).	Ja, när ett projekt byggs så skapas en exe-fil. Exekvering av testscript kan därför göras via en

		fjärrdator genom att exe-filen exekveras. Till filen kan argument anges, se http://www.ranorex.com/support/user-guide-20/lesson-4-ranorex-test-suite.html#c4827.
2. Verktyget har stöd för att exekveringen av ett testfall kan pausas och sedan återupptas.	Ja.	Ja.
3. Rangordning över testfall kan sättas m.h.a. verktyget.	Nej.	Nej.
4. Verktyget möjliggör rollback till initialt läge, om testfall misslyckas.	Nej.	Ja. En teardown region kan anges, vilken exekveras efter ett testfall genomförts eller när ett fel uppstår för ett testfall. I den går det att ange hur applikationen återsätts i normalt läge etc.
5. Verktyget möjliggör exekvering av: inspelade tester, inspelade och editerade tester, samt manuella tester.	Ja.	Ja.
6. Verktyget möjliggör exekvering av inspelade tester för att utvärdera kvalitetskriterier (t.ex. belastningstesting).	Nej, verktyget kan endast utföra funktionell testning.	Nej verktyget kan endast utföra funktionell testning.
7. Verktyget kan ge realtidsfeedback vid scriptexekvering.	Ja, de händelser som utförs visas och statusen för dessa visas i en logg	Ja, de händelser som utförs visas och statusen för dessa visas i en logg
8. Applikationsspecifika karaktäristiker kan testas. I det här fallet har en webbapplikation testats. Följande har undersökts: Crossbrowser testing – möjligheten att testa i flera browser. I det här fallet Internet Explorer, Mozilla Firefox, och Google Chrome	2/3. Mozilla Firefox och Google Chrome ok. Internet Explorer ska gå med tillägg, men det lyckades ej i den här utvärderingen.	3/3. Alla webbläsare fungerar, även om inspelningen i Internet Explorer var lite problematisk och slutade fungera emellanåt.

I Tabell 27 presenteras beslutsunderlag för kriterier som berör testexekvering. Det finns flera nämnvärda skillnader. I Ranorex kan en ”teardown region” skapas, som anger vad som ska utföras efter att ett testfall genomförts, till exempel kan man se till att en applikation sätts i dess utgångläge. Den här möjligheten finns inte i Selenium. När det gäller exekvering från fjärrdator finns det också stora skillnader. I Ranorex går det enkelt att exekvera tester från en fjärrdator genom att köra den exe-fil som skapas när ett testprojekt byggs. För Selenium IDE finns inget stöd för att exekvera tester från en fjärrdator. Genom att använda Selenium RC går det dock (http://docs.seleniumhq.org/docs/05_selenium_rc.jsp). En annan skillnad är att antal iterationer för testfall kan sättas i Ranorex, men inte i Selenium.

Tabell 28. Fånga och jämföra resultat (testexekvering) - beslutsunderlag.

Kriterium	**Selenium**	**Ranorex**
1. Verktyget kan samla in loggningsinformation för exekverade testfall.	Ja, det går även att ange nivå på loggningen (Debug, Info, Warning, Error). Logginformation sparas dock inte till någon fil per default, men det är möjligt genom tillägg.	Ja, det går även att ange nivå på loggningen (Debug, Info, Warning, Error, Success, Faliure). Logginformation sparas per default till en testrapport i mappen för testprojektet.
2. Verktyget kan jämföra faktisk output med förväntad output för ett test.	Ja.	Ja.
3. Testorakel kan skapas m.h.a. verktyget.	Ja, det går att skapa testorakel, men kommandona måste skapas helt på egen hand; d.v.s. det finns ingen inspelningsfunktionalitet för validering. Följande typer av validering kan göras: • Validering av egenskap hos HTML-element, till exempel att en knapp visas, innehåller en viss text, eller har en viss höjd etc. Sker via granskning av DOM:n och måste anges manuellt. Möjliga typer av validering: 1	Ja, det går att validera tillstånd eller egenskaper hos GUI-komponenter vid inspelning av en användarsession. Egenskaper för en GUI-komponenter eller dess bild kan hämtas med hjälp av verktyget. Följande typer av validering kan göras: • Bildvalidering; d.v.s. att det som visas på skärmen vid återuppspelning överensstämmer med den bild som visades vid inspelning. • Validering av egenskaper hos HTML-element. Genom verktygets GUI, går det att ange vilken egenskap som ska valideras, t.ex. att elementet visar en viss text.

		Möjliga typer av validering: 2

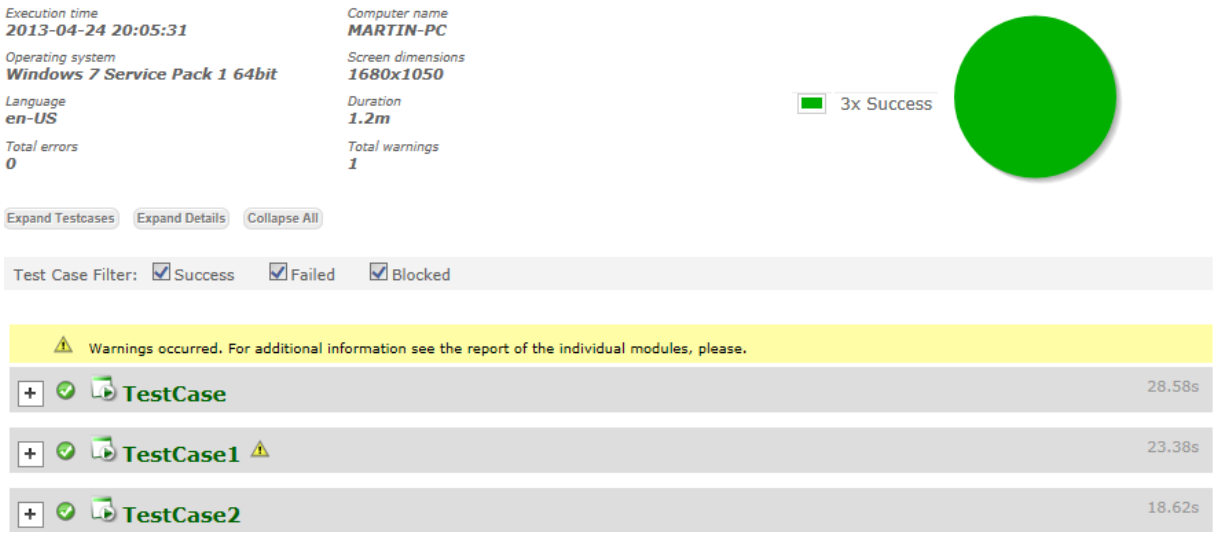
I Tabell 28 presenteras beslutsunderlag för kriterier som berör testexekvering - fånga och jämföra resultat. Det som är värt att nämna är att i Selenium sparas inte logginformation automatiskt till någon fil, men det är möjligt genom tillägg. I Ranorex sker det per default. En annan nämnvärd skillnad är att i Ranorex kan bildvalidering göras. I Selenium går det endast att validera egenskaper hos GUI-komponenter.

Tabell 29. Testrapportering - beslutsunderlag.

Kriterium	**Selenium**	**Ranorex**
1. Verktyget kan ge en sammanställning av logginformation för exekverade testfall (testrapport).	Ej per default, men möjligt med tillägg. De kommandon som körts visas, och även statusen för testfallen, d.v.s. om de lyckats eller ej. Selenium innehåller per default ingen funktionalitet för att spara logginformation till en fil, men det kan fås via tillägg: https://addons.mozilla.org/sv-SE/firefox/addon/file-logging-selenium-ide/?src=dp-dl-othersby. Tillägget FileLogging möjliggör att testrapporter sparas (som XML-filer), även om det måste göras manuellt.	Ja, en testrapport sammanställs automatiskt vid exekvering av ett testfall/en testsvit. Den här testrapporten innehåller information om vilka tester som körts, dess utfall, under vilken tidsperiod, på vilken maskin och operativsystem, vilken fönsterstorlek testerna exekverats i m.m. Rapporten innehåller även detaljerad information om alla testfall, vilka kommandon som utförts, hur lång tid de tagit osv. Testrapporterna visas automatisk på skärmen, och sparas även till en Ranorex-loggfil (.rxlog).
2. Verktyget möjliggör att en inställningsbar mängd testdata *sparas* till en testrapport	Ej per default, men möjligt genom tillägget FileLogging (https://addons.mozilla.org/sv-se/firefox/addon/file-logging-selenium-ide/) går det att ställa in vilken nivå på loggning som ska sparas (Debug, Info, Warning, Error). Antal rapporteringsnivåer: 4.	Ja, det går att ange vilken nivå av data som ska sparas: Debug, Information, Warning, Error, Success, eller Faliure. Antal rapporteringsnivåer: 6. Det går även att ange egen logginformation som ska visas.
3. Verktyget möjliggör att en inställningsbar mängd testdata *visas* för en testrapport	Ej per default, men för tillägget FileLogging går det att välja mellan fyra nivåer att spara till loggfilen: Debug, Info, Warning, Error.	Ja, det går att ange vilken nivå av data som ska visas (se ovan). Det går även att specificera egen logginformation som ska visas.

4. Testrapporter kan automatiskt skickas med hjälp av verktyget till en angiven e-mail adress.	Det finns ingen inbyggd funktionalitet för att skicka mail automatiskt, men det finns beskrivningar kring hur det skulle kunna göras.	Det finns ingen inbyggd funktionalitet för att skicka mail automatiskt. Ranorex har dock utvecklat en kodmodul för det, som kan användas: http://www.ranorex.com/forum/send-mail-module-t2356.html.

I Tabell 29 presenteras beslutsunderlag för kriterier som berör testrapportering. Det finns stora skillnader mellan verktygen. Selenium har ingen rapporteringsfunktionalitet per default, men genom tillägg är det möjligt att spara simpla testrapporter i XML-format. På Seleniums webbsida: http://docs.seleniumhq.org/docs/05_selenium_rc.jsp, beskrivs att rapporteringsfunktionalitet kan fås genom att använda Selenium RC tillsammans med ett ramverk för testautomatisering (exempelvis JUnit). Selenium RC som är en i samlingen av Selenium verktyg, kan alltså integreras i ett ramverk för testautomatisering, och på så sätt kan extra funktionalitet fås. Integration med andra verktyg, och extra funktionalitet genom tillägg är styrkor hos Selenium som är värt att beakta. Tillägg är en betydande faktor hos många open source-verktyg. När det gäller Ranorex finns god rapporteringsfunktionalitet. Loggning till rapporter sker automatiskt per default, och det går att ställa in vad som ska loggas. I testrapporter kan screenshot visas, och en grafisk representation kan ges över exekverade testfall (se Figur 4).



Figur 8. Testrapport i Ranorex.

Tabell 30. Debuggingstöd - beslutsunderlag.

Kriterium	Selenium	Ranorex
1. Verktyget ger stöd för att specificera problem/fel genom att förse användaren med fördefinierade mallar.	Nej.	Nej.
2. Verktyget kan generera noteringar vid fel.	I kommandologen kommer information visas om vilket	Vad som gått fel visas i kommandologen. Fel

	kommando som misslyckats. Felinformation sparas dock inte till en fil.	dokumenteras vanligtvis i testrapporten, det går även ange vilken information som ska visas, t.ex. att testrapporten bara innehåller vad som gått fel.
3. Fel kan prioriteras m.h.a. verktyget.	Nej.	Nej.
4. Verktyget möjliggör övervakning av ändringar på förfrågningar/felnoteringar och dess nuvarande status.	Nej.	Nej.
5. Verkyget kan ge statistisk information över körningar.	Nej.	Nej.

I Tabell 30 presenteras beslutsunderlag för kriterier som berör debuggingstöd. Det skiljer inte så mycket mellan Selenium och Ranorex för den här kategorin.

Tabell 31. Underhållsstöd – beslutsunderlag.

Kriterium	**Selenium**	**Ranorex**
1. Verktyget förser stöd för skötsel av tester (versionshantering, delning och säkerhet etc.).	Ja, Selenium kommer inte med något versionshanteringssystem. Det går dock att få på ett smidigt sätt genom att köra Selenium i Visual Studio eller liknande. Selenium erbjuder att filer genereras i ett antal olika format, vilket gör att det finns stora möjligheter till att ha versionshantering, men för det krävs externa verktyg.	Ja, Ranorex har ett inbyggt gräsnitt mot SVN och TFS.
2. Verktyget kan bidra till att en spårbarhet fås mellan olika element hos testningen, t.ex. mellan logiska och konkreta testfall.	Nej.	Nej.
3. Verktyget kan ge en spårbarhet över ändringar för testobjekt (testscript etc.), och	Nej. Selenium har ingen sådan funktionalitet. Integrationsmöjligheter för att	Nej. Ranorex har ingen egen funktionalitet för det. Dock ger de

kommunicera dessa.	få sådan funktionalitet finns dock, till exempel med Visual Studio.	integrationsmöjligheter som finns (SVN eller TFS) den möjligheten.
4. Verktyget förser stöd vid underhåll av testdata för testfall.	Ja, kommandon anges i form av nyckelord, och det går enkelt att ändra testdatn via drop-down listor.	Ja, kommandon anges i form av nyckelord, och det går enkelt att ändra testdata via drop-down listor. Testdata kan även läsas in från en extern källa, till exempel från en Excel, eller CSV-fil.
5. Verktyget har snapshots-inrättningar (d.v.s. det går att frysa testmiljön i dess nuvarande tillstånd).	Nej.	Ja.

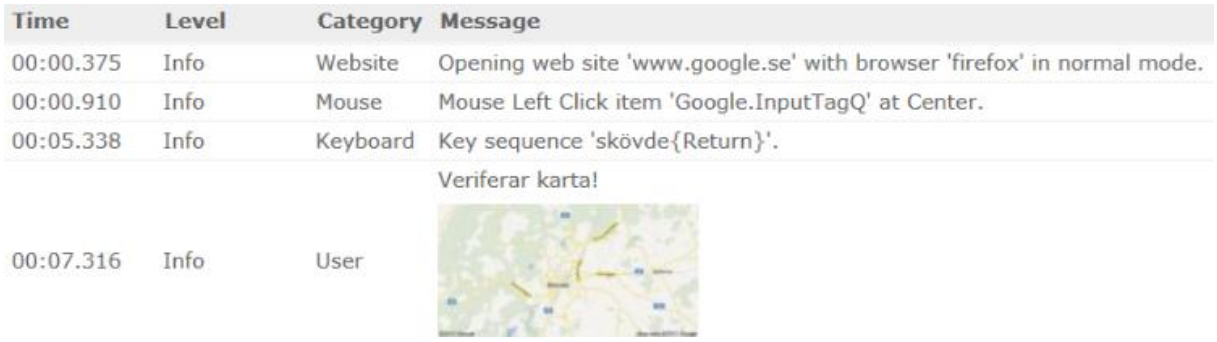
I Tabell 31 nedan presenteras beslutsunderlag för kriterier som berör underhållsstöd. Nämnvärt är att Ranorex har ett gränssnitt mot SVN och TFS för versionshantering av tester. Selenium har inget sådant gränssnitt, men formatet som testscripten kan genereras i ger goda möjligheter till versionshantering. Båda verktygen förser bra stöd för underhåll av testdata. När det gäller snapshot-inrättningar finns endast sådan funktionalitet i Ranorex. De vill säga det går att frysa testmiljön i ett visst tillstånd och vid ett senare tillfälle kan en rollback göras till det tillståndet.

Tabell 32. Kriterier identifierade via praktiken - beslutsunderlag.

Kriterium	**Selenium**	**Ranorex**
1. Felbeteende kan anges för testfall i en testsvit, d.v.s. vad som ska hända när ett testfall misslyckas. [Testexekvering]	Nej, exekveringen kommer alltid fortsätta med nästa testfall i testsviten.	Ja, det går att välja mellan att exekveringen ska fortsätta med nästa testfall, att exekveringen ska avbrytas helt, eller att den ska försöka med samma testfall igen.
2. Antal iterationer för ett testfall kan konfigureras. [Testexekvering]	Nej.	Ja.
3. Tester för verktyget kan exekveras som bakgrundsprocess. [Testexekvering]	Ej hittat information.	Ej hittat information.
4. Uppspelningshastigheten av testfall kan justeras för verktyget.	Ja, det går ange på en skala mellan långsamt till snabbt.	Ja, det går att ange uppspelningshastighet i förhållande till normal

	[Testexekvering]		hastighet (x * normal).
5.	Regular Expressions kan användas i valideringsvillkor. [Testutvärdering]	Ja, se http://docs.seleniumhq.org/docs/02_selenium_ide.jsp.	Ja, se http://www.ranorex.com/blog/ranorexpath-tips-and-tricks.
6.	Screenshots kan visas i testrapporter för verktyget. [Testrapportering]	Nej.	Ja, se Figur 9.
7.	Verktyget kan visa grafiska testdata i testrapporter. [Testrapportering]	Nej.	Ja. Ett cirkeldiagram kan visas över vilka testfall som körts och deras utfall.
8.	Verktyget kan automatiskt ta en bild över det som visades när ett fel uppstod. [Debuggingstöd]	Går inte per default, men är möjligt genom tillägg: https://addons.mozilla.org/sv-se/firefox/addon/screenshot-on-fail-selenium/	Ja, per default tas ett screenshot när ett fel inträffar.

I Tabell 32 presenteras beslutsunderlag för kriterier från fallstudien. Nämnvärt är att i Ranorex kan felbeteende anges, d.v.s. vad som ska hända när ett testfall i en testsvit misslyckas. I Selenium ges inga valmöjligheter – nästa testfall kommer alltid exekveras när fel inträffar. När det gäller testrapportering finns också skillnader. I Ranorex kan grafiskt material visas i testrapporter, i form av cirkeldiagram över utförda tester, samt screenshots. Dessa möjligheter finns inte i Selenium, som inte har någon rapporteringsfunktionalitet per default. Inte heller via tillägg kan det fås, men genom att integrera Selenium i ett ramverk för testautomatisering kan rapporteringsfunktionalitet fås som sagt. Det kräver dock att Selenium RC (http://docs.seleniumhq.org/docs/05_selenium_rc.jsp) används istället för Selenium IDE. Andra skillnader är att i Ranorex tas automatiskt ett screenshot när ett fel inträffar. Det görs inte i Selenium, men möjligheten finns genom tillägg.



Figur 9. Testrapport med screenshot i Ranorex.