CHALMERS

Automatiserade GUI-tester med Selenium

Examensarbete inom Högskoleingenjörsprogrammet i datateknik

MARTIN HELMERSSON
JOSEF HADDAD

Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2014

Automatiserade GUI-tester med Selenium

Martin Helmersson, Josef Haddad

Institutionen för data- och informationsteknik
Chalmers tekniska högskola
412 96 Göteborg
Tel: 031-772 1000
Fax: 031-772 3663

Institutionen för data- och informationsteknik
Göteborg, 2014

Abstract

Software testing is used to ensure the quality of software. The purpose is to verify the system requirements, find bugs and reduce the risk to release a system that contains errors. Large parts of a system can be verified with different tests and test methods. However, it is not possible to test all the different test cases, a fact which is accepted by the software industry.

This thesis aims to develop an automated test suite for the Swedish Tax Agency for registration of crimes, RIF. The test suite should test the graphical user interface. The Tax agency's tester verifies continuous the user interfaces but it takes a long time and is given lower priority by other tests. Tax Agency's claim was that the test tool Selenium would be used.

The work was done in the Swedish Tax agency office in Gothenburg. A test suite was developed with positive results from the Tax agency. Selenium as a tool was evaluated and the possibility of replacing manual testing towards automated. The project demonstrated that it was possible to automate the navigation through the system and manage the system's inputs. Selenium was experienced during the project provides as a simple and powerful test tools. Some parts of the Swedish Tax system is still being checked manually, since the test suite is not evaluated to completely replace the tester.

Sammanfattning

Mjukvarutestning används för att säkerställa kvaliteten på programvara. Syftet med mjukvarutestning är att verifiera systemets krav, hitta buggar samt minska risken att släppa ett felaktigt system. Stora delar av ett system går att verifiera med olika tester och testmetoder. Dock är det inte möjligt att testa alla olika testfall, något som är erkänt av mjukvarubranchen.

Projektets syfte är att utveckla en automatiserad testsvit till Skatteverkets system för brottsanmälan, RIF. Testsviten ska testa det grafiska användargränssnittet. Skatteverkets testare verifierar kontinuerligt användargränssnittet men det är tidskrävande. Skatteverkets krav var att testverktyget Selenium skulle användas.

Arbetet har genomförts på Skatteverkets kontor i Göteborg. En testsvit utvecklades med positivt resultat från Skatteverket. Selenium som verktyg utvärderades samt möjligheten att ersätta manuella tester mot automatiserade. Projektet visade att det var möjligt att automatisera navigationen genom systemet, samt hantera systemets inmatningar. Selenium upplevdes under projektet som ett enkelt och kraftfullt testverktyg. Enskilda delar av Skatteverkets system testas fortfarande manuellt, eftersom testsviten inte utvärderades till att helt ersätta testaren.

Nyckelord: Selenium, GUI, Testning, Skatteverket, RIF

Förord

Rapporten beskriver resultatet av ett examensarbete på 10 veckor (15hp). Arbetet genomfördes på helfart under andra hälften av vårterminen 2014 hos Skatteverket i Göteborg. Examensarbetet ingår som en avslutande del i den treåriga dataingenjörsutbildningen vid Chalmers Tekniska Högskola.

Ett stort tack vill vi först och främst rikta till personalen på Skatteverkets IT-avdelning. Speciellt till Sandra Isgren och Jenny Järpvik, våra handledare på Skatteverket. Under projektets gång har de assisterat och stöttat oss. Den feedback vi fick på vårt arbete bidrog till ökad kunskap om hur det fungerar i arbetslivet. Vi vill också tacka Anders Järkendal som löste inkvarteringen smidigt och såg till att vi kom igång med dator, mjukvara, licenser etc. Stort tack även till vår handledare från Chalmers, Joachim von Hacht, för allt stöd under rapportskrivningen.

Göteborg den 11 maj 2014

Josef Haddad Martin Helmersson

Beteckningar

API	Application Programming Interface. Specifikation på hur mjukvarukomponenter samverkar.
BBT	Black Box Testing. Metod där testaren inte känner till implementationen av modulen som testas.
C	C är ett imperativt och maskinära programmeringsspråk.
C#	C# är ett objektorienterat språk, utvecklat av Microsoft.
C++	C++ är ett objektorienterat programmeringsspråk som bygger på C.
GUI	Grafiskt användargränssnitt (Graphical User Interface).
HTML	Hyper Text Markup Language. HTML används vanligtvis som språk vid skapande av webbsidor.
HTTP	Hyper Text Transfer Protocol. Kommunikationsprotokoll för webbtrafik.
IE	Internet Explorer. Webbläsare från Microsoft.
JSON	JavaScript Object Notation, används för att serialisera JavaScript-objekt via HTTP.
Firefox	Webbläsare från Mozilla.
RIF	Rättsväsendets informationsförsörjning. Omfattande omstruktureringar för rättsväsendets myndigheter.
TDD	Test Driven Development. Utvecklingsmetod där tester skrivs innan implementation.
Visual Studio	Utvecklingsverktyg från Microsoft.
WBT	White Box Testing. Utvecklingsmetod där testaren vet hur moduler är implementerade.
Webbapplikation	Applikation som körs i webbläsaren.
XP	eXtreme Programming, är en systemutvecklingsmetodik som baseras på fem värderingar: *kommmunikation, enkelhet, återkoppling, mod och respekt.*
.NET	Ramverk från Microsoft. Består i huvudsak av en runtimemiljö och ett klassbibliotek.

Innehåll

1 Inledning

1.1 Bakgrund

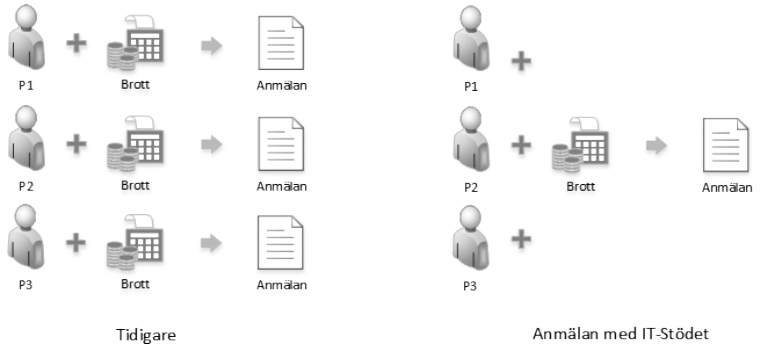
Rättsväsendets informationsförsörjning (RIF) är ett projekt med syftet att utveckla och förbättra informationshanteringen mellan rättsväsendets myndigheter. I en första etapp har ett elektroniskt informationsflöde skapats mellan de myndigheter som hanterar de största ärendemängderna i brottmålsprocessen – Rikspolisstyrelsen, Skatteverket, Åklagarmyndigheten, Ekobrottsmyndigheten, Domstolsverket och Kriminalvården. Arbetet leds och samordnas av Justitiedepartementet [12].

Skatteverket är en förvaltningsmyndighet för beskattning, fastighetstaxering, folkbokföring etc. Verksamheten bedrivs till största del på olika kontor runt om i landet. En viktig uppgift är att ta emot samt granska deklarationer och skatteärenden åt medborgare och företag [10]. På IT-avdelningen i Göteborg arbetar en projektgrupp med Skatteverkets del av RIF. Ett system för att skapa brottsanmälningar elektroniskt har utvecklats. Brottsanmälans väg från Skatteverket till Åklagarmyndigheten sker även den elektroniskt. Innan projektet RIF skickades brottsanmälningar i pappersform mellan myndigheterna. Systemet RIF med elektronisk brottsanmälan driftsattes 2011 [18][19].

Metoden på hur Skatteverket skapar en brottsanmälan har arbetats om i samband med RIF. Innan RIF driftsattes upprättades en brottsanmälan endast av en person, en granskare. Därefter skickades den till Åklagarmyndigheten för utredning. En brottsanmälan innehåller mycket information och många bilagor. För att säkerställa kvaliteten på de brottsanmälningar som upprättas har en metod med tre faser utvecklats. I den första fasen skapas en anmälan av en granskare. I nästa fas kvalitetssäkras anmälan av ytterligare en granskare och i den sista fasen hanterar en beslutare anmälan. En beslutare är en kontorschef eller en särskild förordnad tjänsteman. Figur 1.1 visar förloppet [18][19].

Figur 1.1: *Brottsanmälans faser.*

Innan RIF driftsattes upprättades en brottsanmälan för varje misstänkt person och brott. Med RIF är det istället anmälan som är i fokus. Detta innebär att det är möjligt att koppla flera personer till samma brott och anmälan. Det går även att koppla flera brott till en anmälan. Målet är att minska antalet anmälningar som Åklagarmyndigheten behöver utreda. Figuren 1.2 visar skillnaden mellan föregående metod för brottsanmälan och RIF.



Figur 1.2: *Anmälan i fokus med RIF*

RIF har resulterat i att Skatteverket har utvecklat en webbapplikation för hantering av brottsanmälningar. Webbapplikationen hanterar alla tre faser. Det går alltså att skapa, kvalitetssäkra och besluta om en anmälan i applikationen. Många alternativ kan väljas under upprättandet av en anmälan och det kan bli mycket inmatningar. Med många val och inmatningar tar det lång tid för en testare att testa hela applikationen. Skatteverkets testare verifierar och testar användargränssnittet i webbapplikationen med stickprov, men också med längre sekvenser. Arbetet är tidskrävande och nedprioriteras av andra tester. En grund för att automatisera testerna av användargränssnittet har gjorts med testverktyget Selenium för att påvisa att det är möjligt. På grund av tidsbrist har Skatteverket inte haft möjlighet att slutföra projektet med automatiserade tester. En fördel med automatiserade tester är att de inte tar någon tid från testaren eftersom datorn utför alla tester. Testerna kan därför köras ofta och eventuella fel upptäcks samt åtgärdas i ett tidigt skede. Detta sparar på resurser.

1.2 Syfte

Syftet med projektet är att utveckla en automatiserad testsvit till det grafiska användargränssnittet för Skatteverkets webbapplikation för brottsanmälningar. Målet är att testsviten skall testa alla sidor i webbapplikationen.

1.3 Avgränsingar

Skatteverkets krav är att vi ska använda deras befintliga testverktyg, Selenium. Möjligheten att utvärdera andra testverktyg uteslöts. Endast sekvenser av användargränssnittet kommer att testas.

1.4 Precisering av frågeställning

Tillsammans med projektets mål ska testverktyget Selenium utvärderas. Projektet syftar också till att besvara frågan:

- Kan automatiserade GUI-tester med Selenium helt ersätta en manuell testare?

2 Metod

Metoden vi har använt oss utav är uppdelad i olika steg:

- Instudering av RIF: För att skapa en testsvit till tidigare beskrivet system behöver vi först analysera systemet. Analysen omfattar ingen kodgranskning av systemets uppbyggnad utan endast en genomgång av användargränssnittet.
- Instudering av Skatteverkets Verktyg: Skatteverkets krav var att använda testverktyget Selenium för programmeringsspråket C#.
- Implementation av tester: Efter det påbörjas den iterativa implementationen av tester för att uppfylla kravspecifikationen.
- Testsvit: Testerna knyts ihop till en testsvit som överlämnas till Skatteverket.
- Utvärdering: Efter genomfört arbete utvärderas Selenium som testverktyg och möjligheten att med testsviten ersätta testaren.

3 Teknisk bakgrund

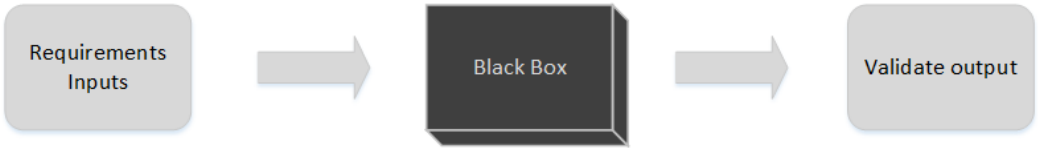
3.1 Testning

Mjukvarutestning används för att säkerställa kvaliteten på programvaran. Syftet är att verifiera systemets krav, hitta buggar samt minska risken att släppa en defekt produkt. Fokus kan ligga på funktionalitet, pålitlighet, stabilitet, användbarhet eller prestanda. Mycket går att testa men dock inte allt, något som accepteras inom mjukvarubranschen. Från en liten betydelse under 1960- och 1970-talet har vikten av tester ökat allt mer. Idag är testning en stor del av mjukvaruutvecklingen [4].

3.1.1 Black Box Testing och White Box Testing

Metoden där testaren inte har någon kännedom om komponentens inre uppbyggnad kallas för Black Box Testing(BBT) [4]. Principen benämns även funktionell testning. När komponenten testas utifrån, utan kännedom om struktur, blir testningen likt ett verkligt scenario. Fördelarna med metoden är att testaren och utvecklaren är oberoende av varandra, eftersom testerna genomförs ur användarens perspektiv. Testerna kan oftast utformas i ett tidigt skede utifrån kraven på systemet. Nackdelen är att vissa tester blir redundanta, eftersom de redan är implementerade på lägre nivåer av utvecklaren. Ofta blir det många testfall och de kan vara komplexa att utforma [4].

Figur 3.1 illustrerar BBT där testaren bara ser utsidan av systemet/programmet. Den svarta komponenten symboliserar systemet eller programmet som ska testas. Data matas in utifrån, i form av inmatningar eller knapptryckningar. Dessa påverkar systemet och en kontroll genomförs på systemets utdata för att se om resultatet blev som förväntat.



Figur 3.1: *Principen för Black Box Testing.*

När en utvecklare skriver tester, ofta för att testa specifika delar av källkoden, benämns det för White Box Testing. Det förekommer även begrepp som Glas Box-, Open Box- och Clear Box Testing. Skillnaden mellan dessa typer och BBT är att testaren har full insikt i systemet och är välbekant med systemets implementation. Till stora system anses båda testmetoderna nödvändiga för att en applikation ska möta kravspecifikationerna [4].

3.2 Enhetstester

Ett enhetstest är ett test som verifierar och testar de minsta enheterna i ett program eller system. Vanligtvis en metod eller en klass. Enhetstester är vanliga när en ny funktion integreras med de övriga i systemet. Vanligtvis är det utvecklaren av funktionen som skriver enhetstesterna. Eftersom enhetstester har en begränsad omfattning behövs även integrationstester göras. Även om alla moduler klarar sin uppgift och samtliga enhetstester går igenom är det inte säkert att de samverkar korrekt. En vanlig typ av integrationstester är BBT där alla funktioner i systemet testas tillsammans. Enhetstestning benämns även modultestning [4].

3.3 Testdriven utveckling

Testdriven utveckling är en av grunderna i eXtreme programming [2]. Med korta cykler, kontinuerlig utveckling samt utvärdering uppnås en snabbare utvecklingsprocess. En cykel består av följande delar;

- Utforma ett test för någon funktion i systemet.

- Implementera en funktion så att den precis klarar testet.
- Omstrukturera och städa upp i koden.

Med testdriven utveckling skrivs testerna före implementationen av koden. Det medför att det är testet i sig som driver utvecklingen av koden och inte en specifikation. Testet blir en form av en specifikation. Utvecklaren kan nu iterativt fokusera på att skriva kod som klarar testerna. Omstrukturering av kod sker löpande i varje cykel tills ett test godkänts. Cykeln upprepas och mjukvaran växer fram [1].

3.4 Agila metoder

Agila metoder har en viktig roll i testdriven utveckling. Under år 2001 samlades flera utvecklare under namnet agila alliansen för att se över metoderna för mjukvaruutvecklingen. Gemensamt för utvecklarna var att de hade tröttnat på den långa och segdragna process som gick ut på att utveckla stora mjukvaruprojekt. Om kunden inte blev nöjd, tog det lång tid att modifiera systemet efter kundens önskemål. Arbetet resulterade till slut i den agila deklarationen (engelska; agil manifesto) vilken innehåller fyra nyckelpunkter för agila metoder:

- "**Individer och interaktioner** framför processer och verktyg".
- "**Fungerande programvara** framför omfattande dokumentation".
- "**Kundsamarbete** framför kontraktsförhandling".
- "**Anpassning till förändring** framför att följa en plan".

- Agila deklarationen 2001.

Medarbetare och arbetslag ska prioriteras under en utvecklingsprocess. Dokumentation tar tid och behöver underhållas. Istället kan en väl skriven och kommenterad kod till viss del användas som dokumentation. Ett gott och nära kundsamarbete är också ett nyckelord. Med korta utvecklingscykler och ett nära samarbete med kund, minskar risken att systemet inte uppfyller kundens önskemål. Den sista punkten speglar att det inte går att planera ett projekt långt in i framtiden. Allt eftersom arbetet fortskrider behövs nya planer. En viktig skillnad mot den traditionella vattenfallsmetoden är implementeringscyklerna. Med en agil metod blir de fler och kortare, till skillnad från en vattenfallsmetod där hela processen är en stor cykel. Scrum och Extrem programmering (engelska; XP programming) är exempel på agila metoder [2][3].

3.5 Microsoft Framework .NET

.NET är ett ramverk av Microsoft som främst körs på operativsystemet Microsoft Windows. Det består av Common Language Runtime (CLR) och ett klassbibliotek. CLR är runtime-miljön som utgör grunden i .NET. CLR exekverar programkoden samt sköter minneshantering, trådhantering och säkerhet. Den förväxlas oftast med Common Language Interface (CLI) som är en specifikation och inte en implementation. Klassbiblioteket är omfattande och har stöd för konsolprogram, grafiska applikationer, webbapplikationer och databaser. Ramverket har stöd för programspråken C#, Visual Basic (VB), C++ och J-Sharp. Visual Studio är utvecklingsverktyget som används för programmering i .NET [11].

3.5.1 C#

C# är ett objektorienterat språk, utvecklat av Microsoft. Språket är en del av plattformen .NET. Officiellt är språket baserat på C++, men det har stora likheter med Java. C# är plattformsoberoende, men utvecklingsverktyget finns endast till Windows. Koden kompileras till bytekod som exekveras i en virtuell maskin. Konceptet med bytekod och virtuell maskin påminner om Java. Språket används till både applikationer för skrivbordsmiljön och för webbapplikationer [6].

3.5.2 ASP .NET

ASP .NET är ett ramverk för utveckling av dynamiska webbapplikationer. Det är uppbyggt på CLR och stödjer samtliga .NET språk. Ett flertal applikationsgrenar finns inom ramverket som Web Forms, MVC, Single-page applications med flera [5].

3.5.3 Web Forms

Web Forms är en struktur för webbapplikationer som bygger på ramverket ASP.NET. När Web Forms presenterades år 2002 låg fokus på att underlätta utvecklingen av webbapplikationer [3]. HTML och HTTP var relativt nya begrepp. Microsoft utvecklade Web Forms med hänsyn till att övergången från Windows Forms applikationer till webbapplikationer skulle bli minimal. Det Microsoft utformade var ett system för hantering av ett klassiskt händelsestyrd grafiskt användargränssnitt via webben. Med Web Forms hanteras samtlig HTML-kod i en markup-fil med filändelsen ASPX. Filen innehåller endast statisk HTML-kod för layout som renderas i webbläsaren. Koden för hantering av event, är flyttad från ASPX-filen till en fil med samma namn men med en annan filändelse. Filen innehåller ingen HTML-kod utan något av .NET-språken C# eller Visual Basic. Detta ger en renare struktur och mer lättläslig kod [4].

3.5.4 Visual Studio

Visual Studio är en utvecklingsmiljö från Microsoft. Miljön hanterar utveckling av grafiska applikationer, telefon-applikationer och webb-tjänster. Det finns också en avancerad testmiljö, Test Editor.

Test Editor är ett verktyg för att hantera enhetstester. Ett enhetstest definieras genom att placera [TestMethod] före metoddeklarationen. Test Editor som innehåller en testvy, vet på så sätt att metoden är ett test. Med testvyn går det att hantera de testmetoder projektet innehåller. Enhetstesterna körs i Test Editorn. Metoder med [TestMethod] hittas av Test Editorn och genom checkboxar väljs de som exekveras. Efter testerna presenteras resultatet grafiskt av en resultat-vy i Test Editorn. Test som inte går igenom och fallerar markeras med ett rött kryss och test som går igenom (blir godkända) markeras med en grön bock [17].

3.6 LINQ

LINQ (Language Integrated Query), är ett API för att hantera förfrågningar till en databas. En förfrågan om data från en databas kallas vanligtvis för; query. LINQ medför att querys kan skrivas för att ändra, ta bort eller hämta information. Dessa i sin tur påminner mycket om tillämpningar som SQL har. LINQ har samma syntax oberoende vilket .NET-språk de skrivs i [9].

3.7 Selenium

Selenium är en samling verktyg för att testa webbapplikationer. Det är öppen källkod. Selenium är kompatibelt med alla webbläsare som har stöd för JavaScript, eftersom det till stora delar är uppbyggt av JavaScript. Selenium är uppdelat i fyra olika verktyg [13];

Selenium IDE	Instickspogram till Mozilla Firefox som kan spela in interaktioner eller händelser. Händelseförloppet kan sedan användas för att generera tester.
Selenium Webdriver	Fjärrstyr olika webbläsare med webbläsarens inbyggda stöd för automation, lokalt eller över nätverk.
Selenium Remote Control	Klient/Server system som möjliggör kontroll av webbläsare lokalt eller över nätverk.
Selenium Grid	Utvidgning av Remote Control. Möjliggör att tester körs på flera servrar samtidigt för att effektivisera och spara tid.

3.7.1 Seleniums uppkomst

Jason Huggins utvecklade år 2004 ett verktyg som kunde utföra automatiska interaktioner på en webbsida. Vid den tiden dominerades marknaden för webbläsare av Internet Explorer. Andra alternativ, främst varianter från Mozilla, började öka i antalet användare. Det var vanligt att utveckling endast var riktad mot Internet Explorer. Det som var speciellt med Jason's verktyg var möjligheten till att styra flera olika webbläsare med samma kod. Det webbläsarna hade gemensamt var möjligheten att exekvera JavaScript. Genom att hela tiden förse webbläsaren med JavaScript som exekverades kunde han automatisera och efterlikna en användares beteende. Verktyget fick namnet Selenium och blev slutligen Selenium Core. [15].

Eftersom Selenium Core var skrivet i JavaScript, medförde det vissa begränsningar. Webbapplikationen som utvecklades och Selenium Core var tvungna att köras på samma maskin på grund av webbläsarens säkerhetsfunktioner. Problemet ligger i att det exekverade JavaScriptet endast kunde göra förfrågningar till samma server som sidan ligger på. Säkerhetsfunktionen är ibland ett problem under utvecklingen och benämns; same host origin". Anrop till andra servrar blockeras av webbläsaren. Lösningen på "same host origin"-problemet var en HTTP-Proxy server. Detta öppnade också en möjlighet att använda ett flertal olika språk. Språket behövde bara klara av att skicka HTTP-förfrågningar till en URL. Utvecklingen ledde till vad som benämns Selenium Remote Control [14][15].

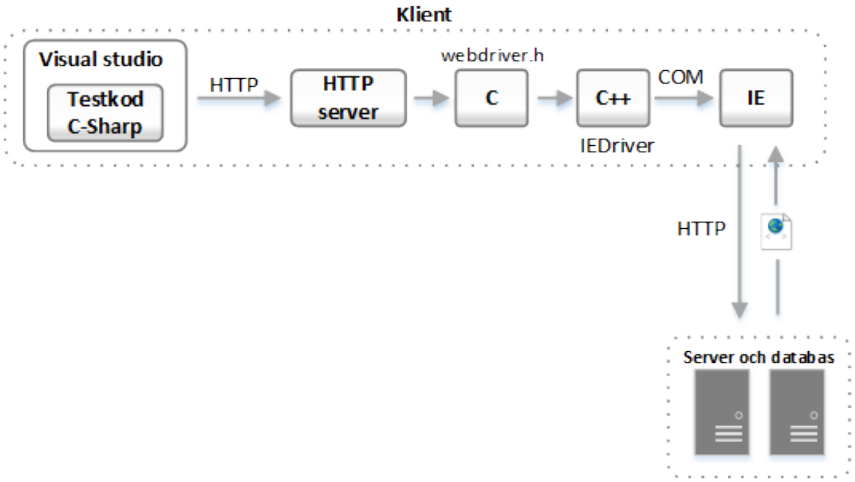
3.7.2 Selenium Webdriver

Webdriver var från början ett eget projekt för att automatisera webbläsare. Projektet hade ingen anknytning till Selenium. Vid lanseringen fanns endast stöd för Java till skillnad mot SRC. Den största skillnaden låg dock i hur de styrde webbläsaren. Selenium Core, grunden till SRC, var en JavaScript-applikation som exekverades inuti webbläsaren. Det Webdriver istället gjorde var att utnyttja webbläsarens inbyggda stöd för automation, detta för att undvika problemet med JavaScript och same host origin". En sammanslagning gjordes av Webdriver och Selenium. Detta resulterade i Selenium 2. I många fall refereras Selenium 2 som bara Webdriver eller Selenium Webdriver. Olika webbläsares har olika stöd för automation och fungerar inte på samma sätt. Därför är Webdriver specifik för varje webbläsare [15].

3.7.3 Internet Explorer Driver

Internet Explorer Driver är Webdrivern som används för att automatisera interaktioner med webbläsaren Internet Explorer. Internet Explorer har ett COM-interface som möjliggör att den kan fjärrstyras. COM är en förkortning av Component Object Model och är ett binärt standard-interface skapat av Microsoft. Det är en modell för kommunikation mellan olika processer och trådar. Interfacet är implementerat på låg nivå och för att kommunicera med testkoden behövs ett mellanlager. Kommunikationen med COM-interfacet sköts med hjälp av klasser implementerade i C++. Funktionerna som kan användas i klasserna finns definierade i en header-fil. För att varje språk inte ska behöva använda metoder för att anropa funktionerna i klasserna med C-kod (Javas - JNA, Pythons ctypes eller C-Sharp:s pinvoke) används en liten HTTP-server. Testarens kod kommunicerar med lokal instans av en HTTP-server via ett JSONWireProtocol över HTTP. Servern anropar i sin tur funktionerna definierade i header-filen. I funktionerna finns kod för att kommunicera med Internet Explorer via COM-interfacet. En översikt över Internet Explorer Webdriver visas i figur 3.2. Alla block på översta raden i figur 3.2 finns på samma maskin som testerna körs ifrån. Det är testkoden som med hjälp av Seleniums Webdriver får webbläsaren att utföra anrop till applikationens värddator [15].

Firefox Webdriver och andra Webdriver skiljer sig på flera punkter från strukturen på IE Webdrivern. Detta på grund av att de är utvecklade av olika företag för olika plattformar [14][15].



Figur 3.2: *Selenium Webdriver för Internet Explorer.*

För att använda WebDrivers funktioner behövs ett antal API:er. Med Webdriver för Internet Explorer är det tre stycken filer som ska importeras till projektet med testkoden. Figur 3.3 visar importen av API:erna i C#. Importen är nödvändig och detta görs med nyckelordet using.

```
using OpenQA.Selenium;
using OpenQA.Selenium.IE;
using OpenQA.Selenium.Support.UI;
```

Figur 3.3: *Import för att använda Webdrivers API.*

Efter importen går det att instantiera ett Webdriver-objekt för Internet Explorer. Objektet är av typen; IWebDriver. En sökväg till HTTP-servern måste anges vid instantiering av ett IWebDriver-objekt. Konstruktorn tar därför en sökväg som inparameter. IWebdriver startar också Internet Explorer. Efter instantieringen är det möjligt att navigera till en sida. Detta görs genom en Navigate-metod följt av en GoToUrl-metod på angivet IWebDriver-objekt.

```
IWebDriver driver = new InternetExplorerDriver("C:\WS\GuiTest\IWebDriver.exe");
driver.Navigate().GoToUrl(ADRESS);
```

Figur 3.4: *Instantiering av IWebdriver.*

För att interagera med sidans innehåll finns IWebElement definierat. Alla HTML-element som button, dropdown, radio med flera går att instantiera som ett IWebElement-objekt. För att hitta dessa, används FindElement-metoden som letar upp elementet via id, link text, XPATH, CSS etc. Därefter kan testerna utföra Click-metoden på elementet för att simulera mustryckningar. Figur 3.5 visar koden som klickar på knappen med id; ”saveContent”. Metoden FindElement behöver veta om det är text, id, eller en länk som den ska söka efter. Det görs med; By.id() [15].

```
IWebElement MyButton = driver.FindElement(By.Id("saveContent"));
MyButton.Click();
```

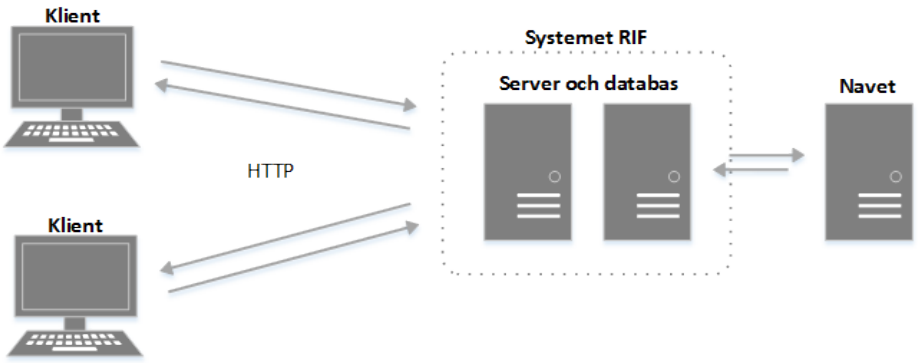
Figur 3.5: *Klicka på knappen med id; ”saveContent”.*

4 Nuvarande system

Webbapplikationen RIF som Skatteverket utvecklar, är en intern webbapplikation. Den har varit i drift sedan 2011. Ny funktionalitet implementeras kontinuerligt av projektlaget som arbetar enligt Scrum [19].

4.1 Övergripande arkitektur

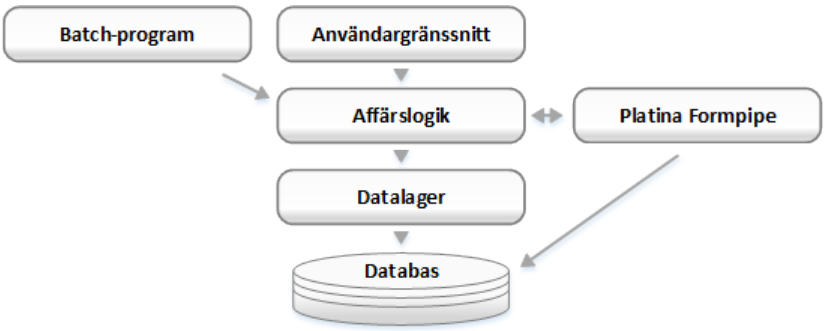
RIF är byggd på Microsofts ramverk för webben ASP .NET Web Forms. Systemet körs från en central server placerad hos Skatteverket. Åtkomst sker genom Skatteverkets interna nätverk. Figur 4.1 visar hur systemet RIF används av olika klienter. Klienterna finns placerade i hela Sverige på Skatteverkets olika kontor. Servern kör applikationen som i sin tur kommunicerar med databasen. I databasen lagras brottsanmälningar som upprättas av klienterna. Personinformation hämtas från Skatteverkets persondatabas; Navet. Systemet behandlar och svarar på förfrågningar från olika klienter. Vid misstanke om brott navigerar en granskare till applikationens webbadress, det grafiska användargränssnittet hämtas till webbläsaren och en anmälan kan upprättas [18][19].



Figur 4.1: *Systemet RIF.*

4.2 Systemets skiktning

Systemet består av flera skikt, vilket ger en tydlig struktur. Målet är att få en inkapslad kod för att skikten skall vara så oberoende som möjligt av varandra. Moduler i samma skikt kommunicerar och känner till varandra. Moduler på ett skikt känner också till moduler på nivån under. Genom att koden är uppdelad blir det lättare att underhålla ett system över en längre tidsperiod. Genom en skiktad struktur kan systemet utvecklas snabbare, eftersom programmera kan jobba med olika skikt samtidigt. Figur 4.2 visar en blockfigur över de olika skikten applikationen består av [18][19].



Figur 4.2: *Skiktningen av systemet RIF.*

4.2.1 Användargränssnitt

Användaren interagerar med det grafiska användargränssnittet. Här sker inmatning, överblick samt verifiering av data som matats in. Figur 4.3 visar välkomstsidan till systemet. Till höger i figur 4.3 visas de olika inkorgarna. ”Mina utkast”innehåller anmälningar som är påbörjade men inte har skickats till en kvalitetssäkrare. ”Mina brottsanmälningar”innehåller anmälningar som skickats till kvalitetssäkrare. Genom att klicka på knappen; Upprätta ny Brottsanmälan, laddas förstasidan för att upprätta en ny brottsanmälan. Sidan visas i figur 4.4.

Figur 4.3: *Välkomstsidan till Systemet RIF.*

Figur 4.4: *Förstasidan till ”upprätta ny brottsanmälan”.*

4.2.2 Affärslogik

Affärslogiken är länken mellan användaren och databasen, det vill säga systemets logik. Skiktet hanterar affärsobjekt i form av personer, anmälningar, lagparagrafer med mera. När användaren interagerar med användargränssnittet görs förfrågningar om data, som hanteras av affärslogiken. Utifrån vad användaren vill se, väljer skiktet ut vilken data som ska hämtas från databasen. Data bearbetas och slutligen når den användargränssnittet där användaren ser resultatet.

4.2.3 Batch-program

Batch-programmet är den del av systemet som skickar de färdiga anmälningarna till Åklagarmyndigheten. Programmet körs automatiskt vid bestämda tidsintervall. Batch-programmet hämtar information med hjälp av affärslogiken och på så sätt kan en brottsanmälan genereras. Brottsanmälan skapas i form av XML-fil. Alla anmälningar som är beslutade och klara, genereras till XML-filer och skickas sedan till åklagarmyndigheten för utredning.

4.2.4 Datalager

Datalagret agerar mellanhand för att komma åt databasen. Den främsta orsaken till att använda ett dedikerat datalager är låg koppling. Genom att lyfta ut databashanteringen behöver bara datalagret ändras om en ny lagringsform skulle vara aktuell istället för att skriva om skiktet med affärslogik. Datalagret använder LINQ [7].

4.2.5 Platina Formpipe

Platina är en produkt från Formpipe som sköter ärendehantering tillsammans med automatiska arbetsflöden. Det görs för att förenkla myndigheternas administrativa arbete. Formpipes tjänster används för att automatisera stora processer hos nästan 90 myndigheter i Sverige. Platina strukturerar informationshanteringen från skapandet till dess att informationen sparas. Platina baseras på lösningar från Microsoft [8].

4.2.6 Databas

Datahanteringen i systemet sköts av en databas av typen SQL-Server från Microsoft. Alla brottsanmälningar sparas av Skatteverket även efter det att de skickats till Åklagarmyndigheten.

5 Genomförande

5.1 Analys

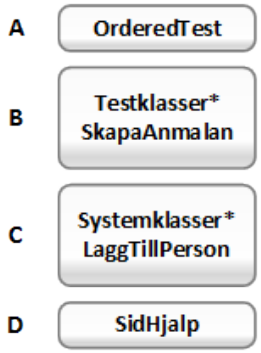
Systemet RIF finns endast lokalt på Skatteverkets nätverk, därför fick vi plats på kontoret i Göteborg. Vi startade med en gemensam analys av deras system för att få en överblick på hur det fungerade. Det gjordes genom att vi framförallt läste en användarmanual, gjord av Skatteverket, samt navigerade själva fritt i systemet. Efter att ha fått en förståelse började vi studera testprojektet vi fick av Skatteverket. Den innehöll en början till ett kort automatiserat test som de hade implementerat. Utifrån detta byggde vi upp våra egna tester med Selenium för att få en större förståelse. Tillsammans med handledarna på Skatteverket gavs punkterna i kravspecifikationen en prioritet. Prioriteringen gick ut på hur viktig del av systemet ett test ansågs vara. På så sätt skulle vi garantera att de viktigaste testerna blev klara i tid. Kravspecifikationen beskriver mer ingående de punkter i systemet som ska testas och finns bifogad som bilaga 1. Tester utvecklades för de olika punkterna i specifikationen. För vissa punkter krävs en sammansättning av flera test. Detta för att verifiera och testa en längre sekvens genom systemet. Testerna utgör tillsammans en testsvit.

5.2 Implementation

Parprogrammering tillämpades till en början eftersom vi båda saknade erfarenhet av C# och Selenium. Under projektet användes inget versionshanterings-system. En anledning till att inget versionshantering-system användes var att Skatteverket ansåg att det inte skulle hjälpa oss. Inlärningströskeln till deras system, Clearcase, ansågs vara hög. Istället fick vi en lokal mapp av Skatteverket där vi kunde arbeta. Successivt gick vi över till att programmera separat samtidigt som vi delade upp arbetet. Med dessa förutsättningar kunde vi arbeta vidare och vid behov föra en dialog om problem uppstod. En stor vikt lades på att försöka få koden väl kommenterad och lättläst, eftersom den kommer att läsas, exekveras samt modifieras av Skatteverket vid ett senare skede. Klasser och metodnamn är namngiva utifrån sina uppgifter. Namngivningen är på svenska eftersom det var ett önskemål från Skatteverkets sida. Systemet RIF använder samma typ av namngivning på metoderna och klasserna.

5.3 Testsvitens design

Testsviten vi har utvecklat är indelad i olika skikt, eftersom samma kod används flera gånger på vissa ställen. Genom en god struktur minskas upprepningar och funktioner kan återanvändas. Kopplingen mellan klasserna blir låg. Figur 5.1 visar en överblick på testsvitens design. Testsviten består av ett antal klasser med olika ansvarsområden. För att enklare förstå klassernas roll har vi delat in de i olika skikt efter deras uppgift. Fyra skikt kan ses i figur 5.1. OrderedTest är placerad överst och innehåller alla test i den ordning de ska exekveras. Går alla test igenom, går även denna igenom (den uppfattas som ett test av Visual Studio). Det finns endast en klass i skikt A; OrderedTest, och endast en i skikt D; SidHjalp. Skikt B och C innehåller flera klasser. Skikt B består av tre klasser med tester, testklasser. En av testklasserna heter; SkapaAnmalan. Det finns en testklass för varje fas i Skatteverkets upprättande av brottsanmälan. Skikt C består av flera systemklasser, som inte innehåller några test. Systemklasserna håller information om den aktuella sidan och har metoder för att interagera med sidan. Ett exempel på en systemklass är; LaggTillPerson, som hanterar sidan där en person utan personnummer knyts till en brottsanmälan.



Figur 5.1: *Testsvitens olika skikt.*
**Flera klasser finns i skiktet.*

5.3.1 Sidhjalp

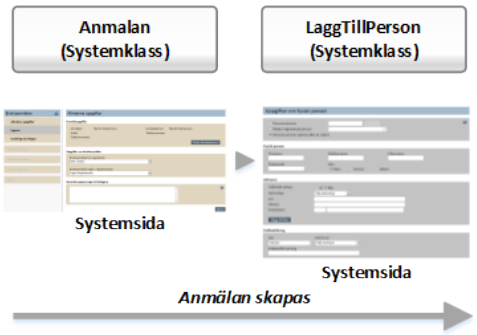
SidHjalp är testsvitens minsta byggsten och den används av alla tester. Varje Systemklass har ett SidHjalp-objekt för interagera med en sida. I SidHjalp finns metoder för att klicka, skicka tecken, välja från dropdown-listor med flera. Dessa metoder används frekvent under alla tester. Metoderna i SidHjalp gör det möjligt att navigera igenom hela RIF systemet. SidHjalp interagerar med sidan genom Seleniums IWebDriver-objekt, tillsammans med olika id:n. Med id i det här fallet menas elementets HTML-id. Varje element på en sida där användargränssnittet visas har ett id. Metoderna som går att anropa i sidhjälp är följande;

- ValjDroppLista(string id, string text)
- Klicka(string id, int timeoutISekunder)
- Klicka(string id)
- KickaPaKnapp(string id)
- KickaPaKnapp(string id, int timeoutISekunder)
- SkrivText(string id, string text, int timeoutISekunder)
- SkrivText(string id, string text)
- PopUpJaEllerNej(boolean yesOrNo)
- HittaElementMedId(string id)
- HittaElementMedId(string id, int timeoutISekunder)
- HamtaTextFranId(string id)
- HittaElementMedXpath(string xpath)

5.3.2 Systemklasser

På nivå C i figur 5.1 återfinns klasser som är relaterade till en specifik systemsida i användargränssnittet. Ett exempel är LaggTillPerson-klassen. Den innehåller alla metoder för att fylla i och knyta en person till en anmälan. Vanligtvis hämtas person- och adressuppgifter med hjälp av personnumret från Navet.

I figur 5.2 visas två systemsidor och de motsvararande systemklasserna. När en sida är testad går testerna vidare med att testa nästa sida. HTML-id:n på elementen som används finns definierade som konstanter i systemklassen för respektive systemsida. Figur 5.3 är ett exempel på hur ett id kan se ut. För att upprätta en anmälan måste flera systemsidor hanteras.



Figur 5.2: *Systemsklasser och respektive systemsida.*

Figur 5.3 visar HTML-kod för en dropdown-lista med id; mpGuide_phContent_Lagrumsgrupp". Genom att anropa metoden; ValjDropLista(string id, string text) i SidHjalp, går det att välja ett alternativ ur listan. Metoden tar dropdown-listans id och texten på de element som ska väljas. Om den andra inparametern är; Skattebrottslagen", väljs det alternativet i listan. Finns inte det angivna alternativet genererar IWebDriver ett undantag.

```
<div class="contentarea">
  <h1>Välj lagrum för misstänkt brott</h1>
  <h3>Lagrum</h3>
  <div class="box gray">
    <a href="#" class="helpiconpopup" onclick="HjalpIkonPopup()"></a>
    Lagrumsgrupp<br />
    <select name="mpGuide$phContent$ddLagrumsgrupp" onchange="javascript:setTimeout(&#39;__doPostBack(\&#39;mpGuic
id="mpGuide_phContent_ddLagrumsgrupp" class="dropdown" style="width:450px;">
      <option selected="selected" value="1">Skattebrottslagen</option>
      <option value="2">Bokf&#246;ringsbrott</option>
      <option value="3">&#214;vriga brottsbalksbrott</option>
      <option value="4">Aktiebolagslagen</option>
      <option value="5">Folkbokf&#246;ringslagen</option>
      <option value="6">Lagen om n&#228;ringsf&#246;rbud</option>
      <option value="9">Utl&#228;nningslagen</option>
      <option value="11">Lagen om straff f&#246;r marknadsmissbruk vid handel med finansiella instrument</option>
```

Figur 5.3: *Exemplel på HTML-kod från Systemet RIF.*

5.3.3 Testklasser

Testsviten innehåller tre testklasser. En för varje fas som Skatteverket använder för att upprätta en brottsanmälan. En testklass är en klass som innehåller testmetoder, där varje testmetod hanteras som ett test. Det finns också fall där flera testmetoder tillsammans utgör ett test, exempelvis OrderedTest. En testklass definieras med ett attribut över klassdeklarationen. Attributet är från Visual Studios Test Editor.

TestClass

Attributet [TestClass] placeras ovanför klassdeklarationen i en klass. Klassen får då Microsofts UnitTest-egenskaper. Den placeras i testvyn och Visual Studio vet att den innehåller testmetoder. Samtliga testklasser innehåller attributet, [TestClass]. Figur 5.4 visar klassdeklarationen av testklassen SkapaAnmalan som hanterar tester för att skapa en brottsanmälan. Samma figur visar även URL:en till startsidan för systemet RIF, i variabeln; Adress. Varje testklass instantierar ett IWebdriver-objekt. Vid instantiering av ett systemklass-objekt skickas IWebDriver-objektet med som parameter till konstruktorn för systemklassen. När systemklasserna sedan använder Sidhjälp, skickas IWebDriver-objektet med som parameter till Sidhjalps konstruktor. IWebDriver-objektet följer alltså med hela vägen till Sidhjalp.

```
[TestClass]
public class SkapaAnmalan
{
    const string Adress =
        "https://nb1130.rsvb.se/platina/Modules/Customized/Rif/Brottsanmalan/Anmalan/Anmalan.aspx";

  public static IWebDriver WebDriver;
```

Figur 5.4: *[TestClass] används i Visual Studio för att definiera att C#-klassen är en testklass.*

TestInitialize och ClassInitialize

[TestInitialize] är ett attribut som möjliggör att vissa metoder exekveras innan varje test. Metoden i figur 5.5, MyTestInitialize, är ett exempel på det. Alla testmetoder definierade i klassen; SkapaAnmalan, utgår från samma URL. Testen startar från början för att sedan arbeta sig djupare in i systemet.

[ClassInitialize()] är ett attribut där det är möjligt att definiera kod som endast exekveras en gång innan samtliga test i en testklass. I testsviten används metoden med [ClassInitialize()] för att plocka ut sökvägen till projektmappen med testsviten. I projektmappen; resources, finns HTTP-servern placerad. Sökvägen behövs när ett IWebDriver-objekt ska instantieras. Efter instatieringen går det att använda API:ets definierade metoder.

```
[TestInitialize]
public void MyTestInitialize()
{
    driver.Navigate().GoToUrl(Adress);
}
```

Figur 5.5: *Testattributet [TestInitialize]*

TestMethod

Attributet [TestMethod] definierar en testmetod. En testmetod utgör ett test. Alla metoder med det attributet placeras i Visual Studios testvy. Metoden VisaSammanstallning_KontrolleraSkickad i figur 5.6 är ett exempel på en testmetod i klassen SkapaAnmalan. Metoden går inledningsvis till en URL, den blåa länken, som leder till applikationens välkomstsida. Ett VisaSammanstallnings-objekt skapas. Objektet definieras i en klass från skiktet under, sidklasser. Vid instantiering av objektet tar den en IEWebdriver genom konstruktorn. Drivern har nu navigerat till den angivna URL:en. VisaSammanstallning börjar navigering ifrån den. I VisaSammanstallning finns ett flertal metoder. En av dessa är KontrolleraSkickad. Metoden hämtar hela sidans innehåll (HTML-koden) och jämför den mot ett antal textsträngar som sidan ska innehålla. Den kontrollerar att anmälan har skickats samt att den har placerats i rätt inkorg. Saknas någon textsträng i sammanställningen eller att den inte överensstämmer med det som matats in, returneras det som saknas eller är fel och testet fallerar. I felmeddelandet går det att se de textsträngar som inte hittades på sidan. Med beskriven metoden går det att validera sidans innehåll i textform.

```
[TestMethod]
public void VisaSammanstallning_KontrolleraSkickad()
{
    driver.Navigate().GoToUrl("https://nb1130.rsvb.se/platina/Modules/Customized/Rif/Brottsanmalan/Valkommen.aspx");
    VisaSammanstallning vSam = new VisaSammanstallning(driver);
    string saknas = vSam.Kontrollera_Skickad();
    if (saknas != null)
    {
        Assert.Fail(saknas + "Saknades under granskning av min brottsanmälan");
    }
}
```

Figur 5.6: *En testmetod med attributet [TestMethod]. Metoden kontrollerar att en anmälan är skickad.*

5.3.4 OrderedTest

OrderedTest är en klass som innehåller samtliga test från alla projektets testklasser. Klassen är inte skriven från grunden utan skapas i Visual Studio med hjälp av ett grafiskt interface. Projektets tester visas i interfacet och det finns möjlighet att själv välja en specifik ordning på hur testerna ska köras. Eftersom testssviten ska kontrollera en anmälans väg genom systemet måste några av testerna exekveras i en viss ordning. Dessutom erhålls tydligare struktur och bättre överblick över testerna. Efter att testerna blivit ordnade i en OrderedTest, hanterar Visual Studio dessa som en testsvit. Efter att testsviten har genomförts med alla tester inkluderade, är det möjligt att se statistik över testsekvensen som genomfördes. Om något test inte gick igenom visas testets felmeddelande, samt vart felet uppkom.

6 Resultat

6.1 Testsvit

Med hjälp av Selenium Webdriver har vi uppfyllt syftet med projektet. Vi har skapat en automatiserad testsvit som testar användargränssnittet enligt given specifikation. Testsviten går igenom hela flödet från det att en anmälan skapas till det att den lämnar Skatteverket. Visual Studio kör och strukturerar våra tester med hjälp av OrderedTest. Det går dessutom att köra enskilda tester för att snabbt se om en viss del av systemet fungerar.

Totalt har 46 testmetoder implementerats och satts ihop till en testsvit. Figur 6.1 visar en vy av testerna från Visual Studio, där samtliga är godkända. Efter att testsviten har kört klart går det att se statistik och körtid för varje testmetod. Högst upp i figur 6.1 visas antalet tester som blivit godkända, i detta fallet samtliga (46 of 46 passed).

Contained tests: 46 of 46 passed

Result	Test Type	Test Name	Owner	Duration
Passed	Unit Test	AnmalanSida_BytKontaktperson		00:00:29.0957011
Passed	Unit Test	KontrollInteSkicka_EjFardigSammanstallning		00:00:08.4733786
Passed	Unit Test	KontrollInteSkicka_EjKanSkicka		00:00:13.7820465
Passed	Unit Test	LaggTillBilagor_LaggTill_Och_TaBort_BrottsBeskrivning		00:00:39.4175450
Passed	Unit Test	LaggTillBilagor_Och_TaBortBilagor_Samtliga		00:00:42.7652534
Passed	Unit Test	LaggTillBrottsmisstanke_KopplaBrottsmisstanke_tillPersoner		00:00:40.5338675
Passed	Unit Test	LaggTillBrottstidpunkterSida_AvtalFaktura		00:00:12.1432947
Passed	Unit Test	LaggTillBrottstidpunkterSida_Bedrägeri		00:00:14.5912600
Passed	Unit Test	LaggTillBrottstidpunkterSida_Betalning		00:00:12.4742777
Passed	Unit Test	LaggTillBrottstidpunkterSida_Bokföring		00:00:12.2992884
Passed	Unit Test	LaggTillBrottstidpunkterSida_Inkomstdeklaration1		00:00:15.9130055
Passed	Unit Test	LaggTillBrottstidpunkterSida_OvrigHandling_Rakenskap		00:00:15.6196442
Passed	Unit Test	LaggTillBrottstidpunkterSida_OvrigHandling_Skattebrott		00:00:13.1157620
Passed	Unit Test	LaggTillBrottstidpunkterSida_PunktskattedeklarationÅrsredovisning		00:00:13.4537649
Passed	Unit Test	LaggTillBrottstidpunkterSida_Skattedeklaration_Moms		00:00:15.0892555
Passed	Unit Test	LaggTillBrottstidpunkterSida_SkattedeklarationMoms_AndraDatumOchUnderlagSaknas		00:00:23.7082555
Passed	Unit Test	LaggTillBrottstidpunkterSida_TaBort_Brottstidpunkt_Skattedeklaration_Moms		00:00:20.6653251
Passed	Unit Test	LaggTillBrottstidpunkterSida_Urkundsforfalskning		00:00:18.6212700
Passed	Unit Test	LaggTillFysiskPersonSida_skapaAnmalanFysiskPerson		00:00:33.5731238
Passed	Unit Test	LaggTillJuridiskPersonSida_LaggTillLagrum_Och_BefintligJuridiker		00:00:15.0522042
Passed	Unit Test	LaggTillJuridiskPersonSida_MedGivetOrgnummer		00:00:07.3382095
Passed	Unit Test	LaggTillJuridiskPersonSida_SkapaNyJuridiker		00:00:11.8117722
Passed	Unit Test	LaggTillJuridiskPersonSida_TaBortJuridiskPerson		00:00:08.2589326
Passed	Unit Test	LaggTillNaringsverksamhetSida_Forsok_TaBortForstaPerson_NRV		00:00:13.4598255
Passed	Unit Test	LaggTillNaringsverksamhetSida_LaggTill_ochTaBort_Mail_Mobil_NRV		00:00:17.1347754
Passed	Unit Test	LaggTillNaringsverksamhetSida_LaggTillFleraPersoner_NRV		00:00:18.9064056
Passed	Unit Test	LaggTillNaringsverksamhetSida_LaggTillPersonUtanPid_NRV		00:00:42.6572464
Passed	Unit Test	LaggTillNaringsverksamhetSida_TaBort_Lagrum_NRV		00:00:14.7602542
Passed	Unit Test	LaggTillNaringsverksamhetSida_TaBortPersonFranLagrum_NRV		00:00:16.8796532
Passed	Unit Test	LaggTillPerson_UtanPersonnummer		00:00:44.3727752
Passed	Unit Test	LaggTillPersoner_MedPersonnummer		00:00:17.9816976
Passed	Unit Test	SkickaInAnmalan_SkickaAnmalan		00:00:54.2008978
Passed	Unit Test	TaBortAllaUtkast		00:02:18.1347628
Passed	Unit Test	ValkomstSidaSkapaNyAnmalan		00:00:02.6983143
Passed	Unit Test	VisaSammanstallning_Kontrollera_Sida_Sammanstallning		00:01:42.6138210
Passed	Unit Test	VisaSammanstallning_KontrolleraSkickad		00:00:05.7778110
Passed	Unit Test	Kvalitetssakra_BytKontaktperson		00:00:06.9830791
Passed	Unit Test	Kvalitetssakra_BytUt_Brottsbeskrivning		00:00:06.3558904
Passed	Unit Test	Kvalitetssakra_Forsok_Kvalitetssakra_Egen_anmalan		00:00:01.8994380
Passed	Unit Test	Kvalitetssakra_Kontrollera_Journalen		00:00:04.2534169
Passed	Unit Test	Kvalitetssakra_KontrolleraNummerOchStatus		00:00:02.5793715
Passed	Unit Test	Kvalitetssakra_LaggTillBilagor		00:00:18.4216126
Passed	Unit Test	Kvalitetssakra_LaggTillLagrum_Och_Juridiker		00:00:10.8557226
Passed	Unit Test	Beslut_Kontrollera_Journal		00:00:03.0577298
Passed	Unit Test	Beslut_Kontrollera_Sammanstallning		00:00:03.2844003
Passed	Unit Test	Beslut_Skicka_Till_Aklagare		00:00:03.0099051

Figur 6.1: *Testmetoder (enhetstester) från testsviten där alla test gick igenom. Bilden är från Visual Studios Test Editor 2010.*

6.2 Utvärdering av Selenium Webdriver

En utvärdering av Selenium Webdriver för Internet Explorer har gjorts efter implementationen av testsviten. Användarvänligheten, möjligheten att ersätta en manuell testare och de problem som uppstod utvärderades.

6.2.1 Användarvänlighet

Vår uppfattning av Selenium Webdriver är att det är enkelt att använda och framförallt komma igång med att automatisera en webbläsare. I vårt fall webbläsaren Internet Explorer. Genom att välja vilket Seleniumverktyg som ska användas och till vilket operativsystem, laddas ett bibliotek ned. Biblioteket innehåller allt som behövs. Det som följer med är filerna som ska inkluderas till projektet och HTTP-servern. Lämpligtvis placeras allt i testsvitens projektmapp. Med de tre raderna från figur 3.3 inkluderas filerna till den aktuella klassen i Visual Studio. Nästa steg är att definiera sökvägen till HTTP-servern. Om datorn har administratörsrättigheter går det att lägga till sökvägen till HTTP-servern i systemvariabeln PATH. Vi hade inte dessa rättigheter på Skatteverkets datorer. Detta löstes med att definiera sökvägen vid instantieringen. Efter att detta är gjort går det att automatisera händelser på valfri hemsida.

Selenium-projektets hemsida erbjuder mycket support med wiki-sidor samt chatt-forum för att stötta utvecklare med frågor. Under projektet användes dessa källor för information om Selenium Webdriver. Mycket finns dokumenterat och ifall något var oklart fanns det oftast personer som ställt frågan tidigare. Sammanfattningsvis var det inga problem att hitta information om hur Selenium Webdriver kan användas.

6.2.2 Möjligheter att ersätta en manuell testare

Skatteverkets projektlag som utvecklar systemet RIF har anställda som enbart utför GUI-tester av systemet. Tester av det grafiska användargränssnittet är mekaniska och tar lång tid. En komplett automatisering av GUI-tester skulle spara tid för testaren och projektlaget. Vår testsvit hanterar inmatningar av data samt verifierar sekvenser genom systemets GUI. Att verifiera sekvenser genom systemet och hantera inmatningar kan testsviten hantera och ersätta den manuella testaren men inte verifieringar av layouten. I alla fall inte som testsviten är uppbyggd idag.

Testspecifikationen i bilaga 1 är skriven utan tanke på layout. Därför blev heller inte testerna designade för att verifiera layout. För att fullständigt ersätta de manuella testerna måste testsviten klara av att verifiera layout också. En person som klickar igenom ett system ser snabbt när ett element ligger fel. Element kanske har fel position och efter våra erfarenheter är detta svårare att verifiera med Seleniums Webdriver. Testsviten ska köras kontinuerligt men vid implementation av nya funktioner kommer manuella tester fortfarande att användas, främst i syfte att kontrollera att sidan ser ut som det är tänkt. Detta görs genom att kontrollera vilka element(knappar, check-boxar, textfält mm) en HTML-behållare innehåller. Sättet att kontrollera vart elementen ligger kräver mycket kod, och tar därför lång tid att implementera. Att tillämpa den verifieringen på systemet RIF var därför inte ett alternativ.

Vid projektets slutskede uppmärksammades en funktion som gör det möjligt för Selenium Webdriver att ta skärmbilder från webbläsaren. Funktionen implementerades aldrig i testsviten på grund av tidsbrist. Istället implementerades ett enkelt test för att ta skärmbilder på en nyhetssida. Ett möjligt scenario är att testsviten tar skärmbilder på systemet under testet. Bilderna på systemet kan sparas i en mapp, som en testare kan gå igenom. Den tid det tar för testaren att klicka igenom hela systemet kan då sparas. Eftersom testsviten går igenom flödet med alla sidor behöver testaren endast granska bilderna. Den manuella testaren behövs fortfarande men genom att granska bilder efter testet kan tid sparas. Figur 6.2 visar koden för metoden som tar en bild på innehållet webbläsaren visar. Koden förutsätter att ett IWebDriver-objekt har definierats som driver".

```
public void TakeScreenshot()
{
    try
    {
        Screenshot ss = ((ITakesScreenshot)driver).GetScreenshot();
        ss.SaveAsFile(@"D:\Screenshots\SeleniumTestingScreenshot.jpg", System.Drawing.Imaging.ImageFormat.Jpeg);
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
        throw;
    }
}
```

Figur 6.2: *C#-Kod för att ta en skärmbild.*

6.2.3 Problem med Selenium Webdriver

En webbapplikation tar olika lång tid att laddas beroende på hur belastat nätet samt servern är. Ett problem vi hade med Webdriver var att sidans innehåll inte alltid hann laddas klart. Testsviten letade efter element som inte hade laddats klart. Detta resulterar i att komponenter på sidan inte hittas och Webdriver kastar exception (testkoden slutar exekvera och testet fallerar). Testet fallerade men vi visste inte om det berodde på att det var fel i systemet RIF eller om HTML-objektet inte hade laddats klart i webbläsaren. Lösningen på problemet var att frysatesttråden. Genom att låta tråden sova medan sidan laddas försvann problemet. Väntetiden kunde vara olika lång beroende på vart vi befann oss i processen. Men vi hittade en tillräckligt lång väntetid som gjorde att sidan alltid hann laddas klart innan något utfördes (knapptryckning etc.). Ibland kunde det ta många sekunder innan sidan svarade, oftast när mycket information hämtades. Detta är inte hållbart i längden, eftersom testtråden alltid sover lika länge. Oavsett hur lång tid det tar för sidan att laddas. Det gör att testerna tillsammans kan ta längre tid.

Webdriver har en wait-metod som ska lösa detta problem men av någon anledning fick vi inte den att fungera till vårt system RIF. Vi började undersöka om det fanns ett annat sätt. Vi hittade en lösning på ett programmeringsforum som var att exekvera ett JavaScript som kontrollerar HTML-taggen readyState. JavaScriptet returnerar olika statusar beroende på om sidan fortfarande laddas eller om den har laddats klart. Om hela sidan har laddats klart returneras ”completeäv JavaScriptet. Dock räcker det inte alltid att kontrollera readyState-taggen utan tråden måste ibland sova ändå. En webbapplikation med mycket JavaScript som hämtar innehåll dynamiskt ändrar inte taggen eftersom taggen ändras bara med request av typen; HTTP-get. Med JavaScript är det möjligt att hämta innehåll och uppdatera en liten del av sidan. Detta är svårare att kontrollera dynamiskt med Webdriver. Däremot var lösningen mycket bättre än föregående lösning, samt minskades testkörningstiden ytterligare.

Lösningen med att exekvera JavaScript som kontrollerar statusen på HTML-objektets readystate-tagg fungerar inte i alla system. Med en Single-page application hämtas sidan endast en gång inledningsvis med ett HTTP request. Genom JavaScript hämtas innehåll efterhand asynkront eller när användaren interagerar med användargränssnittet. Nya knappar kan läggas till efter att ett nytt HTML fragment hämtats via HTTP och JSON. Eftersom inget nytt HTTP-request har skett med JavaScript, sker även ingen uppdatering av HTML-objektet. HTML-objektets readystate”är alltså oförändrad. Vid avläsning är den “complete” och då försöker Selenium hitta knappen som inte existerar. Ett undantag(exception) genereras och exekveringen avbryts. En implementation där testkoden och tråden sover blir nödvändig.

7 Diskussion och slutsats

7.1 Diskussion av resultat

Selenium Webdriver har visat sig innehålla flera effektiva lösningar och är dessutom användarvänlig. Det är möjligt att testa stora delar av innehållet på en webbsida. Allt från att skriva till en textruta till att analysera CSS-layout. Utifrån det har vi skapat automatiserade tester som körs mot Skatteverkets system. Med Visual Studios Test Editor har vi ordnat testerna med; OrderedTest. Utan den möjligheten hade vi varit tvungna att lägga ner mycket mer tid på att försöka ordna dem på ett annat sätt. Vi känner oss nöjda över att ha levererat fram en produkt enligt kravspecifikationerna.

På frågan om det går att ersätta en manuell testare med Selenium Webdriver så beror det givetvis på vilket system som ska testas. I fallet med RIF gick det inte att ersätta testaren för att kontrollera layout. Även om kod hade implementerats för att ta skärmbilder hade ändå bilderna behövts studeras i efterhand av en testare. Även om testaren inte kan ersättas, går det att spara mycket av testarens tid.

7.2 Problem

Första veckan på Skatteverket gick till stor del åt att skaffa behörigheter till diverse program, främst utvecklingsverktyget Visual Studio. Vi upplevde det som negativt eftersom vi nu hade en vecka mindre på oss att hinna bli klara. Nu i efterhand känner vi oss ändå nöjda. Tiden användes till att läsa användarmanualen för att kunna navigera igenom deras system.

Skatteverket har restriktioner på vad som kan laddas ner samt vilka sidor som får besökas via deras nätverk/datorer. Det begränsade projektets möjligheter till att använda ett versionshanteringssystem. Det stora problemet vi fick till en början var när båda hade gjort modifieringar i samma klass och sedan skulle lägga upp det i den lokala mappen. Det blev lätt hänt att vi missade små saker och det tog lång tid innan vi kunde hitta alla ändringar. Problemet hade vi i någon vecka innan vi sedan delade upp arbete på ett sätt där vi inte ändrade i samma klasser längre. Så fort ändringar i en fil var gjorda placerades den på den delade mappen så båda fick tillgång till den nya koden.

7.3 Intervju

Vi intervjuade projektledaren på Skatteverket som svarade på ett par frågor. Frågorna skulle ge oss en större inblick i hur deras testare jobbar idag samt hur han tror att automatiserade tester kommer att hjälpa dem i framtiden. Enligt handledaren, sitter båda deras testare i nuläget och testar förhand. När en ny funktion läggs till i deras system, testar de bara den nya funktionen. Anders uttrycker det så här:

> *”De har genomfört punktinsatser mot ställen i systemet där de vet att förändringar har skett.”*

En anledning till varför automatiserade tester inte har implementerats tidigare är att de alltid haft det i baktanke men brist på tid har lett till att det skjutits fram hela tiden. Samtidigt har det prioriterats bort med tanke på att det tar tid att göra samt underhålla de.

> *”Vi kommer hitta de där små felen vi i vanliga fall inte hittar, för vi vill alltid ha ett så felfritt system som möjligt. Att höja kvaliteten på produkter och hitta fel så tidigt som möjligt är av stor vikt. En annan sak är att det sparar en massa tid samt att det blir mer ekonomiskt.”*

Detta är svaret vi fick på frågan om effekten och hur automatiserade tester skulle hjälpa dem. En viktig sak tas även upp, nämligen att hela RIF-projektet håller på en lång tid framöver, minst till år 2020. Det

skulle innebära att våra automatiserade tester kommer att hållas vid liv en lång tid framöver och vara till stor nytta.

7.4 Kritisk Diskussion

Om vi skulle göra ett liknande projekt idag skulle vi främst vilja ändra på en sak, att använda oss utav ett versionshanterings-system. Med hjälp av den skulle vi spara en hel del tid, och även ha möjlighet att gå tillbaka till tidigare versioner. Vi skulle även kunna jobba i två olika delar utan att få en massa konflikter. Dessutom hade vi enkelt sett vilka förändringar som skett.

7.5 Miljöaspekter

En fördel med automatiserade tester är att de kan köras automatiskt vid en bestämd tid. Därför kan testsviten, som kan ta ett par timmar, köras på natten när det är låg belastning på nätverket och elnätet.

7.6 Slutsats

Både vi och Skatteverket är nöjda med resultatet. En presentation om testsviten och Selenium gjordes för samtliga projektgrupper på Skatteverket. Det upplevdes som väldigt positivt bland alla närvarande. I efterhand har tre andra projekt inom Skatteverket varit i kontakt med oss. Vi fick kolla i deras system och uppskatta hur lång tid vi tror att det tar för dem att bygga automatiserade tester till sina system. Alltså finns ett stort intresse för automatiserade tester. Dessutom är detta något dem verkligen behöver och kommer att ha nytta utav.

8 Referencer

8.1 Litteratur

[1] Cohen, M *Succeeding with Agile* (2010) Boston: Pearson Education

[2] Dooley, j. (2011) *Software Development and Professional Practice.* New York: Apress

[3] Freeman, A. (2013) *Pro ASP.NET MVC 5.* Fith edition. New York: Apress

[4] McWherter, J and Hall, B (2010) *Testing ASP.NET Web Applications.* Indianapolis: Whiley Publishing

8.2 Elektroniska

[5] ASP.NET Overview. Besökt 30 maj 2014, från: http://msdn.microsoft.com/en-us/library/vstudio/4w3ex9c2(v=vs.100).aspx

[6] C Sharp (programming language). Besökt 28 maj 2014, från: http://en.wikipedia.org/wiki/C_Sharp_(programming_language)

[7] Creating a Data Access Layer (C#). Besökt 30 maj 2014, från: http://www.asp.net/web-forms/tutorials/data-access/introduction/creating-a-data-access-layer-cs

[8] Formpipe, Platina. Besökt 30 maj 2014, från: http://www.formpipe.com/sv/Produkter/Formpipe-Platina/

[9] LINQ (Language-Integrated Query). Besökt 28 maj 2014, från: http://msdn.microsoft.com/en-us/library/bb397926.aspx

[10] Om Skatteverket. Besökt 2 juni 2014, från: http://www.skatteverket.se/omoss/omskatteverket.4.65fc817e1077c25b832800015922.html

[11] Overview of the .NET Framework. Besökt 29 maj 2014, från: http://msdn.microsoft.com/en-us/library/zw4w595w.aspx

[12] Rättsväsendets informationsförsörjning (RIF). Besökt 20 maj 2014, från: http://www.regeringen.se/sb/d/1912/a/135442

[13] Selenium Webdriver (Developer article). Besökt 28 maj 2014, från: http://aosabook.org/en/selenium.html

[14] Selenium Projects. Besökt 28 maj 2014, från: http://docs.seleniumhq.org/projects/

[15] Selenium Documentation. Besökt 28 maj 2014, från: http://docs.seleniumhq.org/docs/01_introducing_selenium.jsp

[16] Software testing. Besökt 27 maj 2014, från: http://en.wikipedia.org/wiki/Software_testing

[17] Unit Test Basics. Besökt 30 maj 2014,
från: http://msdn.microsoft.com/en-us/library/hh694602.aspx

8.3 Muntliga

[18] Isgren, Sandra. Utvecklare projekt RIF. Intervjuad 21 maj 2014
sandra.isgren@skatteverket.se

[19] Järkendal, Anders. Projektledare RIF. Intervjuad 25 maj 2014
anders.järkendal@skatteverket.se

[20] Järpvik, Jenny. Utvecklare projekt RIF. Intervjuad 21 maj 2014
jenny.järpvik@skatteverket.se

9 Bilaga 1

Step Name	Description	Expected
Step 1	Anmälaren väljer att misstänkt brott är begånget av fysisk person. Kontrollera att det går att lägga tillflera misstänktapersoner enligt SK06 och att uppgifterna sparas Hämta från Navet Egenskapad Redan registrerad Flera e-postadresser Flera telefonnr	
Step 2	Medverkansform, enligt SK07	Flera misstänkta personer kan läggastill ochsparas
Step 3	Kontrollera att det går att ta borten misstänktperson	Det går att ta bort en misstänkt person
Step 4	Kontroller att det inte går att tabort den första misstänkta personen då brott begånget inom NRV	Den första misstänkta personen gårinte att ta bort

Step Name	Description	Expected
Step 1	Kontrollera att man som anmälare kanvälja att upprätta en ny brottsanmälan	Handläggaren kan välja att skapa enny brottsanmälan
	Välj att skapa en ny anmälan.	
Step 2	Kontrollera att systemet visar rättuppgifter, enligt SK02	Rätt uppgifter visas
Step 3	Kontrollera att det går att byta kontaktpersonoch att rätt uppgifter visas för kontaktpersonen	Det går att byta kontaktperson ochrätt uppgifter visas för kontaktpersonen
	A. Kontrollera att man kan välja:	
	A1. utredningsform för upprättande av brottsanmälan, enligt SK03	A1. Utredningsform kan väljas enligtSK03
	A2. riksplansinsats	A2. Riksplansinsats kan väljas
Step 4	B. Kontrollera att särskild upplysningtill åklagaren kan anges	B Upplysning kan skrivas
Step 5	Kontroller att det går att välja lagrumenligt SK04 och att lagrumstext visas för valt lagrum	Lagrum går att välja enligt SK04 ochlagrums textvisas för valt lagrum
Step 6	Här ska testfallen för juridiker, fysikeroch NRVköras	
	Kontroller att det går att lägga tillflera underlag och tidpunkter enlige SK08 och att systemet genererar tidpunkternakorreket.	
	Ta bort brottstidpunkter	
Step 7	Använd olika inlämningslistor	Flera tidpunkter och tidpunkter kan läggas till, enligt SK08 att tidpunkterna blir korrekta
Step 8	Kontrollera att specifikationerna blir korrekta	Specifikationen blir korrekta
Step 9	Kontrollera att brottsmisstankar skapas (misstänkte person knyts till brottstidpunkten)	Brottsmisstankar skapas
Step 10	Kontrollera att brottsinformation och brottsmisstankeinfomation går att ändra i avancerat läge	Brottsinformation och brottsmisstankeinfomation går att ändra i avancerat läge
Step 11	Kontrollera att bilagor kan läggastill enligt SK11	Bilagor kan läggas till
	Kontrollera att brottsbeskrivning kan läggs till.	
	Mall	
Step 12	Egenskapad	Brottsbeskrivning kan läggas till
Step 13	Kontrollera att systemet validerar brottsanmälan korrekt, enligt SK16	Systemet validerar brottsanmälan
Step 14	Kontrollera att sammanställnigen serkorrekt ut enligt AF015	Sammanställnigen ser korrekt ut
	Kontrollera att:	
	verksamhetsgren måste anges enligt SK23	Verksamhetsgren måste anges
	val av region	Region måste anges
	val av kvalitetssäkrare	Kvalitetssäkraremåste anges
Step 15	meddelande till kvalitetssäkrare kan anges	Medddelande kan skrivas
	Kontrollera att systemet	
	skapar ett ärende med ärendenummer	Systemet har skapar ett ärende med ärendenummer
	att status sätt till "För kvalitesttäkring"	Status är sätt till "För kvalitetssäkring"
	att ärendet är flyttat till rätt inkorg	Ärendet är flyttat till rätt inkorg
	att journalen uppdateras	Journalen är uppdaterad
	att diarieföring är gjord	Diareiföring är gjord
	att meddelande visas, att ärendet ärskickat för kvalitetssäkring	Meddelande visas att ärendet är skickat för kvalitetssäkring
Step 16	att välkomssidan visas	Välkomssidanvisas

Step Name	Description	Expected
Step 1	Anmälaren väljer att misstänkt brott är begånget inom juridisk person.	
	Kontrollera att juridisk person kan läggas in ianmälan enligt SK05 och att systemet spararuppgifterna om juridiken Hämta från BasInfo Egenskapad	
Step 2	Redan registrerad	Juridisk person kan läggas till enligtSK05 ochuppgifterna sparas
Step 3	Kontrollera att juridisk person kan tas bort	Juridisk person kan tas bort

Step Name	Description	Expected
	Skapa en anmälan som innehåller brott begånget inom: Juridisk person Fysisk person	
Step 1	NRV	Anmälan går att skapa
Step 2	Kontrollera att det går att kvalitetssäkra anmälan	Anmälan kan skickas för beslut
Step 3	Kontroller att anmälan kan beslutas	Anmälan kan skickas till annan myndighet

Step Name	Description	Expected
Step 1	Anmälaren väljer att misstänkt brott är begånget av fysisk person.	
	Kontrollera att det går att lägga tillflera misstänktapersoner enligt SK06 och att uppgifterna sparas Hämta från Navet Egenskapad Redan registrerad Flera e-postadresser Flera telefonnr	
Step 2	Medverkansform, enligt SK07	Flera misstänkta personer kan läggastill ochsparas
Step 3	Kontrollera att det går att ta borten misstänktperson	Det går att ta bort en misstänkt person
Step 4	Kontrollera att anmälarenkan lägga till flera misstänkta personer vid lagrumsgruppen "Skattebrottslagen"	Flera misstänkta personer kan läggastill