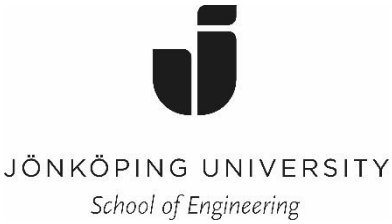


Utveckling av testverktyg för mjukvara med fysiska komponenter

HUVUDOMRÅDE: *Datateknik*
FÖRFATTARE: *Adam Andersson*
HANDLEDARE: *Vladimir Tarasov*
JÖNKÖPING 2018 November

Detta examensarbete är utfört vid Tekniska Högskolan i Jönköping inom Datateknik. Författaren svarar själva för framförda åsikter, slutsatser och resultat.
Examinator: Anders Adlemo
Handledare: Vladimir Tarasov
Omfattning: 15 HP (grundnivå)
Datum: 2018-11-06

Postadress:	Besöksadress:	Telefon:
Box 1026	Gjuterigatan 5	036-10 10 00 (vx)
551 11 Jönköping		

Abstract

A lot of time is spent on manual software testing. When software can interact with exponentially more and more components the number of tests required will increase at the same rate. Automation of these manual tests has been previously shown that a time-saving can occur with test automation.
When tests need to be done on software that interacts with physical components the results of testing automation are harder to predict. This thesis aims to explain the impact of physical components on the development and the result of test tools developed for use on automated tests on objects with physical components.
In this work, a test tool for automatic regression testing for height adjustable table legs were developed.
The method used to design the tool was Design Science Research where, for example, TESLA was adapted for use in automated tests with physical components. TESLA is a language for specifying test cases and allowing test automation with embedded systems that was implemented in the test tool's design which could be used to automatically generate scripts and automatically executes test cases. With the development and design of the testing tool, the thesis attempted to answer how a test tool should be designed when physical components are included in the test object and how the measurement accuracy of the tool may affect the test result.
Experiments have been carried out during the development process where regression tests have been performed according to a test protocol. During these experiments a test protocol is executed. One manually by hand and one is done automatically with the testing tool. The results of the experiments show that the execution of the test protocol execution with the test tool gave a time saving of 35% compared to the manual tests. The physical components of the test object were shown to affect the automatic execution time negative and further analysis of previous research has shown that this result is insufficient to justify the large initial time that is required to automate tests.

Keyword - Test Automation, Physical Components, ROI, Regression Testing, Time Saving

Sammanfattning

Mycket tid spenderas ofta på manuell testning av mjukvara. Då en mjukvara kan interagera med exponentiellt mer komponenter så kommer antalet tester att öka i samma takt. Automatisering av dessa manuella tester har med tidigare forskning bevisat att en tidsbesparing kan ske med testautomatiseringen.
Då tester måste utföras på mjukvara som interagerar med fysiska komponenter så är resultaten av testautomatisering inte lika klara och arbetet syftar på att förklara de fysiska komponenternas påverkan på utveckling och resultatet för testverktyg som används till automatisk testning av testobjekt med fysiska komponenter.
I detta arbete så framtogs ett testverktyg för automatiska regressionstest för höj och sänkbara bordsben.
Metoden som användes för att utveckla verktyget var Design Science Research där till exempel TESLA anpassades för användning i automatiska tester med fysiska komponenter. TESLA är ett språk för att specificera testfall och möjliggöra testautomatisering i inbyggda system implementerades i testverktygets design och kunde då automatisk generera skript och automatiskt exekvera testfall. Med utvecklingen och design av testverktyget så försökte arbetet svara på hur ett testverktyg bör vara designat när fysiska komponenter ingår i testobjektet och hur testverktygets mätnoggrannhet kan påverka testresultatet.
Experiment har utförts under arbetets gång där regressionstester har utförts enligt ett testprotokoll. Under experimenten utfördes ett testprotokoll manuellt för hand och ett automatiskt med testverktyget. I båda situationerna så mättes testprotokollets utförande i tid. Testverktyget gav en tidsbesparing på 35% visavi de manuella testerna. Testobjektets fysiska komponenter hade en negativ påverkan på den automatiska exekveringstiden och vidare analys av tidigare forskning har visat att detta resultat inte räcker för att motivera den stora initiala tidsinvestering som krävs för att automatisera tester.

Nyckelord – Testautomatisering, fysiska komponenter, ROI, regressionstestning, tidsbesparing

Innehållsförteckning

1 Introduktion

Arbetet utfördes tillsammans med ROL ERGO, ett företag som tillverkar kontorsmaterial och är en av världens största leverantörer av elektroniska höj-och sänkbara bordsstativ. ROL:s behov av att förbättra sina regressionstester för deras höj och sänkbara bordsstativ ledde till ett framtagande av ett automatiskt verktyg för regressionstester och kunskaper och idéer om hur design och utvecklande av ett liknande system bör vara.

1.1 Bakgrund

Testning är en mycket viktig del inom utveckling av mjuk/hårdvara, utan kunskaper om testernas betydelse i utvecklingen så kan detta leda till projektledare som ser tester som ett kostsamt och tidskrävande problem som kan skapa konflikter mellan utvecklarna och testare. Tester borde enligt Halili [1] ses som en investering i kvalitet, manuell testning har tidigare visats kunna ge en ROI upp till 350% [2] då resurser som används till kvalitétstester återhämtas genom att undvika omarbetande i projektets senare livscykel i form av buggar och fel som måste fixas. Automatisering av tester syftar på att ytterligare förbättra ROI för testningen. Tidigare sågs automation av tester som en överflödig investering men har nu blivit en nödvändig vidareutveckling inom testning för att den manuella testningen inte räcker till när applikationer och system blir större och mer komplexa [3]. Testningen behöver då effektiviseras genom automatisk testning där repetitiva och tidskrävande testgenomföranden utförs med hjälp av t.ex. skripts som exekverar testfall automatiskt. Automatisering av tester kräver en större initial investering och kan i fall som till exempel i Mohacsi och Beers projekt [4] visa på att då det kan krävas mer än en dubbelt så stor investering i automatisk testning innan den första testcykeln kan genomföras. Den initiala investeringen syftar till att löna sig i de senare testcyklarna genom att effektivisera testningen. Automatiseringen av testerna ledde i Mohacsi och Beers projekt till att exekveringstiden minskade med 91% och tid spenderad på underhåll minskade med 89% per testcykel gentemot manuell testning [4].

I fall där mjukvara som websidor, program eller inbyggda realtidssystem (Real-Time Embedded Systems, RTES) testas så finns det mycket färdiga verktyg för automation som tester som till exempel automatiskt mäter responstider på en websida eller simulationer där tusentals testiterationer kan köras under några millisekunder. Det blir mer komplicerat då mjukvarans funktioner ska testas i samband med fysiska objekt som påverkas av till exempel elektriska motorer och sensorer. Då manuella mjukvarutester utförs tillsammans med fysiska objekt så måste funktionerna exekveras i realtid vilket inte bara leder till att testaren får mycket tidskrävande och repetitiva arbetsuppgifter utan även att den mänskliga faktorn ökar vilket kan leda till fler fel [5]. Automatisering av dessa tester kan minska den mänskliga faktorn och effektivisera testets exekveringstid. Minskandet på testernas exekveringstid har tidigare bevisats [4] men bevis på att samma testmetoder ger liknande resultat när fysiska objekt interagerar med mjukvaran behöver beprövas.

Enligt Stockdale [6] så är regressionstestning är en av de första tester som automatiseras av företag som startar testautomatisering. Regressionstestning kan utföras manuellt och automatiskt och dess repetitiva tester passar bra för automatisering och kan frigöra tid till testarna [7]. Regression i mjukvara är en oväntad förändring som uppstår vid kodändringar eller när nya funktioner implementeras. För att förhindra dessa regressioner så behöver mjukvarans grundläggande funktionalitet testas, då antalet kodändringar ökar så behöver regressionstesterna öka i samma takt. För att ge en idé om hur regressionstesterna ska vara utformade så ställer Woody [7] i journalen ”better software” frågorna “What if a customer asked you to demonstrate to him, within an hour, that your newest software is ready for use? What tests would you run?”. Woody syftar då på att en tumregel för de testfall som väljs i regressionstestning ska ses som bevis att testobjektet kan utföra de grundläggande funktionerna som kunden kräver.

I artikeln skriven av Thoss, Beckmann, Kroeger, Muenchhof och Mellert [8] så beskrivs ett exempel på en automatisering av regressionstester för datoriserad numerisk styrning (Computerized numerical control, CNC) till fräsmaskiner. Artikeln beskriver ett hybrid testramverk baserat på flera teknikdomäner som Eclipse, TPTP (En testmodells plugin till Eclipse) och MATLAB. Detta testramverk används för att manipulera testobjektet för regressionstesterna och visar arkitekturen på ett verktyg samt hur testerna evalueras och

rapporteras. Automatiseringen av CNC-tester gav mindre falska fel jämfört med de manuella testerna och ökade antalet, hastigheten och pålitligheten av regressionstesterna. Detta projekt är ett exempel på utveckling av ett verktyg som används för att testa mjukvara med fysiska komponenter. Testfallen omfattar här färdiga funktionen i CNC styrningen vilket inte svarar på hur exekveringen ska ske om testobjektets inte har färdiga funktioner för test. Testfallen skriptas manuellt och besvarar inte heller hur dessa testfall skriptas. All input till CNC omfattar ej fysiska komponenter och sker via en PC, den enda fysiska komponent i testobjektet är fräsmaskinen vilket endast omfattar utdata ifrån testobjektet. I fall där även testobjektets input sker via fysiska komponenter så kan detta ge annorlunda resultat och behöver utforskas.

För att automatisera exekveringen av tester så används ofta skriptning. Skriptning kan användas för att utföra de funktioner som krävs för att manipulera och verifiera testobjektet. Användning av script-språk som Python [9], Perl [10] och Bash [11] kan skapa skripts som används i testautomation. Dessa skript kan användas för att manipulera, anpassa och automatisera anläggningarna i ett befintligt system [12] och ger mindre komplexa program med kortare och mer snabbskriven kod då de skrivs med en hög abstraktionsnivå [13].

Två teststrategier som kan användas i manipulering av ett testobjekt är White och Black-boxtestning.
White-boxtestning är en strategi där testning bygger på de interna path:s, strukturen och implementationen av systemet testas. Testningen omfattar en undersökning mellan alla path:s i strukturen och att mjukvarans moduler exekveras korrekt [14]. När endast mjukvara testas så är White-boxtestning en bättre anpassad teststrategi som kan ge en djupare insikt i mjukvaran.
Black-boxtestning är då en strategi som är mer anpassad för fysiska system då den ignorerar de interna detaljerna och modellerna av systemets beteende och fokus ligger på att testa hela systemets funktionalitet baserat på objektets krav och specifikationer [14]. Testmetoden syftar till att testa testobjektet genom att påverka objektet via indata och observera det utdata som förväntas. Indata i fysiska system kan till exempel vara knappar, reglage och touch-interfaces och utdata kan till exempel vara temperatur i ett klimatsystem eller bestå av position för en CNC maskin. Då black-boxtestning kan appliceras på praktiskt taget alla system i dess olika utvecklingsfaser så fungerar detta i enhets-, integration-, system- och regressionstester vilket omfattar alla nivåer av testning [15].

1.2 Problembeskrivning

När mångfalden produkter i ett uppkopplat system ökar, så ökar också kombinationerna av produkter som kan interagera med varandra. Med fler möjliga kombinationer så ökar komplexiteten inom systemets delar och resulterar i mer oförutsägbara fel som uppstår under utveckling [16]. För att bekämpa detta behöver systemet testas flera gånger kontinuerligt under utvecklingen och mycket tid kan spenderas på manuell regressionstestning som präglas av mätfel, mycket på grund av den mänskliga faktorn. Då automatisk testning har många fördelar gentemot manuell testning så har ROL ERGO, ett företag som utvecklar intelligenta höj och sänkbara arbetsplatser, behov av att automatisera sina manuella tester för sina höj och sänkbara bordsstativ.
Tester av mjukvara och RTES är lättare automatiserade då det finns flertal simulationer och färdiga verktyg som kan användas. Problem uppstår då dessa interagerar med fysiska system som kräver anpassade verktyg och testsystem. Tidigare forskning visar på teorier om testautomatiseringens fördelar [2], [4], [17] och hur utvecklingsprocessen från manuell till automatiseringen av tester bör ske [18].I dessa fall så har de endast tillämpats inom mjukvarutester eller RTES tester och inte när både indata och utdata sker via fysiska komponenter.

1.3 Syfte och frågeställningar

Syftet med examensarbetet är att undersöka hur ett verktyg för automatisk skriptning och exekvering av testfall bör vara utformat för att kunna utföra och godkänna regressionstest i för black-boxtestning tillsammans med fysiska testobjekt för att öka tidsbesparingen och mätnoggrannheten under testexekveringen.

För att besvara syftet har det brutits ner i 3 frågeställningar.

1. Hur bör ett verktyg för automatisk skriptning och exekvering av regressionstester vara utformat när fysiska komponenter ingår i ett testobjekt?

   Automatisk testning är ett väl utforskat område med många teorier om hur automatisk testning ska exekveras och implementeras. Hur kan dessa teorier implementeras i automatisk skriptning och exekveringen, hur påverkar de resultatet och fungerar de som tidigare bevisat när fysiska komponenter ingår i testobjektet?

2. Vilken tidsbesparing kan uppnås vid användandet av ett verktyg för automatisk skriptning och exekvering visavi manuell testning?

   Automatiska tester har tidigare bevisats ge stora tidsbesparingar under exekvering [4]. Kommer tidsbesparingen vara liknande när fysiska komponenter ingår i testobjektet?

3. Vilken skillnad i mätnoggrannhet kan uppmätas vid användandet av ett verktyg för automatisk skriptning och exekvering visavi manuell testning?

   Vilka metoder kan användas för att utvärdera tester med verktyget och hur skiljer de sig med de manuella testmetoderna.

1.4 Omfång och avgränsningar

Experimenten kommer endast omfatta regressionstest av en typ handset och en typ bordsstativ. Arbetet kommer däremot att analysera delar utanför detta system för att göra antaganden om existerande och framtida delar i systemet för att anpassa verktyg inför framtida vidareutveckling.

Figur 1 - IDRIVE BASIC HANDSET

Handset som används i detta arbete är IDRIVE BASIC HANDSET. Detta handset användes då den har ett simpelt interface med 2 knappar och har mer komplexa funktioner utöver höjning och sänkning så som att sätta max och minimumhöjd för bordsstativet.

Figur 2 - EXPRESS STAND(vänster), modifierad EXPRESS STAND(höger)

Bordsstativet som användes i detta arbete är en modifierad ”EPI 2-column 2-stage (650) Circular” bordsstativ där bordsbenen sitter tätt intill för att minska arbetsytan.

Arbetet kommer framförallt att fokusera på skriptning och utvecklande av verktyg för exekvering av färdiga testfall. Även om automatisk generering av testfall kan ses som en viktig del inom automatisk testning så kommer detta inte att granskas explicit då det anses vara för stort omfång för tidsperioden. Automatisk generering av testfall kommer däremot att tas hänsyn till under verktygets utveckling då detta måste beaktas för framtida utveckling för att få ett mer fulländat testsystem.

1.5 Disposition

Rapporten är utformad efter den standardiserade mall skapad av Jönköping University för examensarbete och används på Jönköpings Tekniska Högskola.

Kapitel 1 - *Introduktion*
: Ger en bakgrund för studien och problemet som behandlas, presenterar även studiens syfte, frågeställningar och disposition.

Kapitel 2 - *Metod och genomförande*
: Beskriver först studiens arbetsprocess i helhet och därefter beskriver arbetsprocessens delar i detalj och vilka metoder som används.

Kapitel 3 - *Teoretiskt ramverk*
: Behandlar den teoretiska grund och förklaringsansats som krävs för studien och det syfte och frågeställningar som formulerats.

Kapitel 4 – *Empiri*
: Presenterar de empiriska erfarenheter och den empiriska data som skapades under arbetets gång.

Kapitel 5 - Analys
: Svar på studiens frågeställningar genom behandling av insamlad empiri och teoretiskt ramverk.

Kapitel 6 – *Diskussion och Slutsatser*
: Diskussion och slutsatser om arbetets resultat, implikationer och begränsningar samt förslag till företag och om vidare forskning

2 Metod och genomförande

2.1 Koppling mellan frågeställningar och metod

Den huvudsakliga forskningsmetoden i arbetet är utförandet av en Design Science Research (DSR).

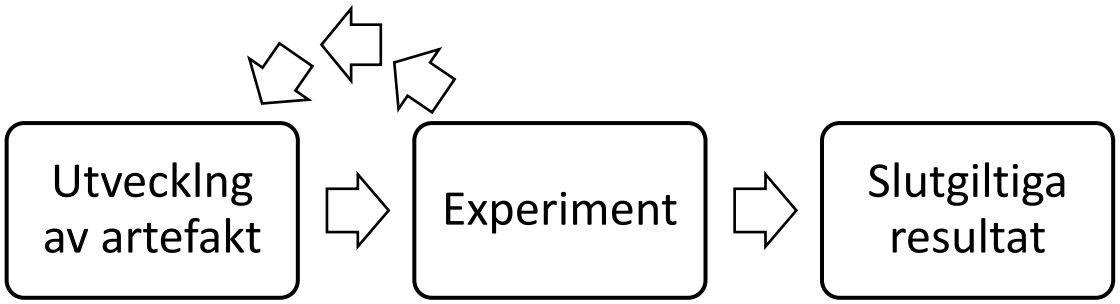
En iterativ forskningsmetod efterfrågades för arbetet då en iterativ utvecklingsprocess är en viktig del av mjukvaruutveckling. Två forskningsmetoder, DSR och Aktionsforskning är båda forskningsmetoder med iterativa strukturer [19] [20]. Dessa forskningsmetoder delar vissa egenskaper som till exempel utföranden av experiment och en resultatfokuserad forskning men skiljer sig på många sätt [19]. Då arbetet syftar sig vara mer allmänhetsgiltig i problemställningen med ett fokus på all mjukvara med fysiska komponenter så passar DSR bättre som forskningsmetod. Aktionsforskning syftar på att studera ett mer idiografisk (mindre generellt) område än DSR där studien på att även ge Nomotetisk (mer generellt) kunskap med utvärderingen av resultatet [19]. DSR lägger också ett större fokus på konstruktionen av en artefakt vilket kan hjälpa att besvara första frågeställningen.

Resultatet av DSR kommer framtas och verifieras genom experiment som utförs iterativt med utveckling. Genom iterativa experimenten så är idéen att detta inte bara kommer att ge insikt i arbetets frågeställningar till slutet av arbetsprocessen, utan även att det ger kontinuerliga resultat som kan analyseras för att gynna utvecklingen och jämföras med varandra för att get en helhetsbild av arbetets utveckling.

Med Design Science Research så kommer arbetet svara på den första frågeställningen ”Hur bör ett verktyg för automatisk skriptning och exekvering av regressionstester vara utformat när fysiska komponenter ingår i testobjekt” genom att implementera idéer så som teorier, funktioner, modeller och metoder relaterade till testområdet i utveckling av verktyget. Med experimenten så kommer verktyget utvärderares och analyseras för att ta reda på vilka idéer som anses bidragit till resultatet och de brister i verktyget och testerna som behöver vidareutvecklas. Arbetet kommer med hjälp av de bidragande teorierna ge en idéer om hur detta verktyg bör vara designat.
Under experimenten så kommer tidtagning ske för att jämföra tidsskillnaderna emellan manuella och automatiska tester för att besvara den andra frågeställningen ”Vilken tidsbesparing kan uppnås vid användandet av ett verktyg för automatisk skriptning och exekvering visavi manuell testning?”. Den tredje frågeställningen ”Vilken skillnad i mätnoggrannhet kan uppmätas vid användandet av ett verktyg för automatisk skriptning och exekvering visavi manuell testning?” kommer besvaras av analys av de manuella mätmetoderna gentemot verktygets mätmetoder.

2.2 Arbetsprocessen

Arbetsprocessen är uppdelad i flera delar, se figur 4. Starten av arbetsprocessens omfattar utvecklingen av artefakten som ska testas genom experiment. Utvecklingen av artefaktens första version kommer ske genom DSR där litteraturstudier och analys av testprotokoll kommer bidra till utformandet av den hårdvara och mjukvara som krävs för att genomföra ett experiment. Resultaten ifrån experimenten analyseras och används i nästa fas för att vidareutveckla artefakten inför nästa experiment. Arbetet syftar på att minst utföra en iteration av utveckling följande experiment och ett avslutande experiment. Ytterligare iterationer med utveckling följande experiment baseras på den återstående tid för examenarbetet.



Figur 3 - Arbetsprocessen

2.3 Ansats

Arbetet kommer använda sig av både induktiva och deduktiva ansatser enligt Blomkvist och Hallin [21] för att ge svar på de frågeställningar som arbetet presenterar. Användandet av TESLA 3.4 är ett exempel på en deduktiv ansats för att besvara hur verktyget bör vara designat. Den tidigare vetenskapliga modellen används då för att anta problemet. Hypotesen att denna teori kommer bidra till verktyget kommer verifieras eller deduceras genom experiment.

En induktiv ansats innebär enligt [21] att man är öppen för att de empiriska resultaten ger en annan teoriram än den initiala. En induktiv ansats används för att besvara hur verktyget bör vara designat, med den iterativa cykeln med utveckling av artefakt och utförande av experiment så förväntas experimenten visa brister i de teorier och idéer som verktyget baseras på för att dra slutsatser om hur verktyget ska förbättras.

2.4 Design

Arbetet kommer utforma sig efter genom Design Science Research, artefakten i detta arbete är ett verktyg för automatisk skriptning och exekvering av regressionstester. Hypotesen är att artefakten(verktyget) ska kunna utföra testsituationer i enlighet med de manuella testerna. För att utveckla artefakter så kommer arbetet följa de fem huvudaktiviteter inom DSR presenterade av Enligt P. Johannesson och E. Perjons [20].

Figur 4 Metodram för DSR enligt P. Johannesson och E. Perjons [20]

Den första aktiviteten, ”Explicera problem” anger delen av DSR där det praktiska problemet analyseras och undersöks för att visa att problemet inte bara är signifikant för en lokal praxis utan att det även finns en signifikans för allmän praxis. Relaterat till arbetet så omfattas denna aktivitet under problembeskrivningen 1.2 där ROL ERGO:s lokala problem att automatisera tester med höj och sänkbara bord generaliseras för att ge problemet en global signifikans inom testautomatisering. Aktiviteten ”Definiera krav” syftar på att ge en översikt på lösningen av problemet i form av krav för artefakten. Under rubriken ”analys av testfall" så ges exempel på hur denna aktivitet används för att analysera ett testprotokoll eller testobjekt för att definiera kraven för hårdvara/mjukvara och för artefakten.

De tre sista aktiviteterna representeras i arbetsprocessen för arbetet, aktiviteten ”Design och Utveckling av artefakt” är självförklarande och representerar själva utvecklingen av verktyget inför experiment. ”Demonstration av artefakt” och ”Utvärdering av artefakt” syftar på att bevisa artefaktens validitet och hur väl den förhåller sig till de fördefinierade kraven vilket är i

linje med experimentets syfte. Experimenten passar bra till utvärderingen av artefakt då de syftar på att utföra en situation som verktyget ska användas i och därmed utvärderar artefaktens funktioner. Då arbetet syftar på att skapa en iterativ utföra experiment och utveckling så påpekar P. Johannesson och E. Perjons [20] att DSR nästan alltid är en iterativ process och den sekventiella metodramen sett ovan endast ska tolkas som de mest betydande av indata och utdata emellan aktiviteter, en iterativ process är naturligt med DSR och passar i arbetet då utvecklandet av verktygets viktigaste syfte är att testas genom experiment.

2.5 Datainsamling

Huvudsaklig datainsamling kommer ske genom experiment. Experimenten utförs genom att genomföra en situation där ett testprotokoll bestående av flera testfall utförs för att verifiera funktionaliteten för testobjektet. Testfallen omfattar instruktioner om hur testobjektet ska manipuleras och testobjektets förväntade resultat. Testprotokollet utförs normalt manuellt av en utvecklare eller testare som följer ett testprotokoll skrivet i ett Excel-dokument. Testfallen utvärderas under exekvering och markeras PASS (godkänd) eller FAIL (misslyckad) i samma Exceldokument som senare blir testresultatet.
I de iterativa experimenten så kommer testfallen utföras likt de manuella där exekvering sker via testverktyget. Ett initialt experiment kommer utföras där testprotokollet kommer exekveras manuellt av författaren för att bestämma manuella exekveringstiden för testprotokollet. Fyra manuella genomföranden av testprotokollet kommer att utföras och mätas med tidtagarur i en mobil för att få en genomsnittlig exekveringstid. När verktyget anser kunna utföra en exekvering av testprotokollet godtyckligt till de manuella testerna (samma testresultat) så kommer experiment utföras automatiskt. Dessa automatiska experiment ska utföras iterativt två eller flera gånger under hela utvecklingsprocessen där dessa också ska utföras fyra gånger per experiment för att få en genomsnittlig testexekveringstid.
Noll hypotesen för de automatiska experimenten är att de ingen tidsbesparing kan finnas med automatiseringen, då att det manuella utförandet av testprotokollet är snabbare. De manuella experimenten kommer anses som kvasi-formella då författare utför de manuella testerna och anses då ha annorlunda insikt i testfallen än vid normal testning och kommer att ge kvantitativa data i form av testexekvering och kvalitativa data i form av personlig analys under detta experiment. De automatiska experimenten kommer anses som formella då de endast kommer bidra med kvantitativa data i form av exekveringstid och testresultat.

2.6 Dataanalys

Data som genereras av experimenten i form av testens exekveringstid. Denna empiriska data bearbetas genom att beräkna medelvärde för att ge ett pålitligt resultat så att manuella och automatiska testsituationernas mätdata kan jämföras. Variabler som Med jämföranden så kommer resultaten analyseras för att motivera de eventuella skillnaderna som uppstår.

2.7 Trovärdighet

För att ha en hög trovärdighet i arbetet så använder sig arbetet forskning relevanta till testning i problematiseringen och ge en hög validitet. Arbetet har även försökt att använda sig av mer nyligen publicerade referenser för att använda sig av en mer ”state of the art” fakta och teorier då forskning inom mjukvara och hårdvara utvecklas konstant och delar av äldre forskning kan snabbt ses som föråldrade och minska validiteten på studien. Experimenten som utförs under arbetets gång anses ha en hög validitet då tidsbesparing för testautomatisering anses vara data som är mycket viktig för ge att motivera testautomatisering och ge idéer om hur fysiska komponenter kan påverka tidsbesparingen. Även fast arbetet skedde på en relativt liten skala så anses experimenten ändå ha en hög extern validitet då den fortfarande ger ett exempel på en procentuell tidsbesparing med testautomatisering för mjukvara med fysiska komponenter.

3 Teoretiskt ramverk

3.1 Koppling mellan frågeställningar och teori

För att ge en teoretisk grund för den första frågeställningen (se nedan) så beskrivs följande områden under rubrikerna Regressionstestning, Skriptning, TESLA och Bergmarks samanställda krav på lyckad testautomatisering.

> *Hur bör ett verktyg för automatisk skriptning och exekvering av regressionstester vara utformat när fysiska komponenter ingår i ett testobjekt?*

För att ge en teoretisk grund för den andra frågeställningen (se nedan) så beskrivs området under rubriken Regressionstestning och Bergmarks samanställda krav på lyckad testautomatisering.

> *Vilken tidsbesparing kan uppnås vid användandet av ett verktyg för automatisk skriptning och exekvering visavi manuell testning?*

För att ge en teoretisk grund för den tredje frågeställningen (se nedan) så beskrivs området under rubriken Sensorer.

> *Vilken tidsbesparing kan uppnås vid användandet av ett verktyg för automatisk skriptning och exekvering visavi manuell testning?*

3.2 Regressionstestning

Ett flertal testtyper används för att testa mjukvara under utvecklingstiden. Funktionell testning är en typ av testning som används för att testa specifika beteenden(funktionaliteter) i ett system. Tester som att söka flygfärd i ett bokningssystem eller testning av skydd mot skadliga attacker mot en applikation anses som funktionalitetstester. Mjukvaran testas här för att se att den beter enligt de tidigare specificerade funktionskraven och utförs ofta tidigt i utveckling i samband med programmering av dessa funktioner. Strukturell testning sker i stilen White-boxtestning, det interna systemet testas och dess kopplingar kontrolleras för att kontrollera mjukvarans programstruktur.

Syftet med regressionstestning är enligt Dustin, Rashka, Paul [22] är att verifiera att åtgärder som sker för att fixa programvaran inte i sin tur skapar ett nytt fel i ett annat område i mjukvaran. Regressionstestning sker i stilen black-boxtestning och där tester utförs för att testa mjukvarans grundläggande krav. Utförandet av testerna sker närstan alltid innan ”release”, för att få en så stabil mjukvara som möjligt så måste dessa regressionstester även utföras så ofta som möjligt under hela livscykeln för mjukvaruutvecklingen för att förhindra regressioner. Då regressionstester repeteras frekvent så är de väl passande för automation och kan ge en hög ROI om automatiseringen kan minska den tid som krävs till manuell regressionstestning under testcykeln likt fallstudien presenterad av Graham och Fewster [4].

3.3 Skriptning

För att förstå vad skriptning innebär så behövs en förståelse av skillnaden med programmeringsspråk och skriptsspråk. Enligt Crowder [23] så är ett sätt att skilja script-språk gentemot andra programmeringsspråk att script inte behöver kompileras innan exekvering. Program skrivna i programmeringsspråk som c och Java måste kompileras innan

körning och kan anses inte som script-språk medan script-språk som JavaScript och Pearl inte behöver kompileras. Crowder påpekar att denna tolkning om vad ett script-språk är och inte är har blivit mer suddigt de senaste åren. Moderna kompileringstekniker som till exempel Googles V8 [24] som är en JavaScript-motor som kan kompilera JavaScript till maskinkod snabbt under exekvering har minskat skillnaden på ett script-språk och andra programmeringsspråk. Dessa tekniker i kombination med modern hårdvara förklarar varför tolkningen av vad ett script-språk är har försvårats. Python [9] är ett annat programmeringsspråk som fortsätter att komplicera grupperingen av script-språk. Python anses av många vara ett script-språk då Python har en hög abstraktionsnivå och verkar kunna köras utan kompilering. Om ett script-språk inte behöver kompileras innan exekvering så anses Python inte vara ett script-språk då Python kod kompileras innan exekvering men gör detta ”on the fly”.

Skriptning är då användandet av dessa ”script-språk” för att utföra en uppgift som till exempel kontrollering av en applikation och ger en mer snabbskriven kod då script-språken ofta har hög abstraktionsnivå.

3.4 TESLA

I konferenspublikationen så beskriver Wahler, Ferranti, Steiger, Jain och Nagy [25] TESLA, ett språk för att specificera testfall för testning av inbyggda system. Testspecifikationer skrivs ofta i ett naturligt sätt och presenteras i ett format läsbart av människor till exempel i form av Word eller Exceldokument. Eftersom ett naturligt språk ofta är mångtydig och saknar struktur så passar de ej för automatiserad exekvering. Målet med TESLA var att specificera ett språk som kunde minska feltolkningarna av testfallen, förkorta testspecifikationen och möjliggöra automatisk testexekvering som fortfarande liknar provspecifikationer som använder naturligt språk.

För att möjliggöra testautomatisering så använder sig TESLA av Action block och Checkblock. En simpel förklarning av ”blocken” är att action-block syftar på att manipulera ett testobjekt till ett visst tillstånd och checkblock används för att verifiera att detta tillstånd uppstod på korrekt sätt och att det tillstånd som uppstod är samma det förväntade tillståndet beskrivet i testfallet.

Tabell 1 TESLA checkblock enligt Wahler, Ferranti, Steiger, Jain och Nagy [25]

Check type (Kontrolltyp)	Beskrivning
In-interval check	Verifiera att ett påstående är sant minst en gång under ett intervall
Never-in-interval check	Verifiera att ett påstående förblir falskt under ett intervall
Change-to-true-in-interval check	Verifiera att ett påstående initialt är falskt men blir sant under det specificerade intervallet.
Duration check	Verifiera att ett påstående blir sant under ett intervall och förblir sant under en viss tid innan den blir falskt.

Verifieringen av testobjektets tillstånd kan ske med olika tidskrav och kan ses i *Tabell 1*. Varje check specificerar ett tidsintervall $[t_{start}, t_{slut}]$ och ett eller fler påståenden som kontrolleras. Om påståendet ej uppnås under dessa tidskrav så anses kontrollen misslyckad och därmed testfallet misslyckat.

In-interval check

$\exists t \in [t_{start}, t_{slut}] \quad P(t)$ [25]

Denna kontroll ser att om predikat P någon gång under tiden t inom tidsintervallet $[t_{start}, t_{slut}]$ är sant så är kontrollen är korrekt.

Never-in-interval check

$\forall t \in [t_{start}, t_{slut}] \quad \neg P(t)$ [25]

Never-in-interval check är tvärtom In-interval check där tiden t under tidsintervallet $[t_{start}, t_{slut}]$ så är predikatet P alltid falskt.

Change-to-true-in-interval check

$\exists t \in [t_{start}, t_{slut}] \left(\forall t' \in [t_{action}, t-1] \quad \neg P(t')\right) \cap P(t)$ [25]

Change-to-true-in-interval check är en kombination av båda de tidigare beskrivna kontrollerna, två påstående testas under denna kontroll. Det första påstående är att under tiden t som är en del av tidsintervallet $[t_{start}, t_{slut}]$ så är predikat P sant. Det andra påståendet är behöver P vara falskt under den initiala tiden t' under intervallet $[t_{action,} t-1]$ där t_{action} är tiden då ett actionblock utfördes. Kontrollen är endast godkänd om båda dessa påståenden är sanna inom tidsperioden.

Duration check

$\exists t \in [t_{start}, t_{slut}] \quad \left(\forall t' \in [t_{action}, t-1] \quad \neg P(t')\right) \cap \left(\forall t'' \in [t, t+d-1] \quad P(t'')\right) \cap \neg P(t+d)$ [25]

Duration check har tre påståenden som alla måste vara sanna för att en kontroll ska vara godkänd. Under tiden t' som är en del av tidsintervallet $[t_{action}, t-1]$ så måste predikat P vara falskt. När predikat P senare blir sant under tiden t'' i intervallet $[t, t+d-1]$ så måste predikat P vara sant under hela tidsintervallet för att senare bli falskt efter den bestämda varaktigheten.

3.5 Bergmarks samanställda krav på lyckad testautomatisering

I studien Utvecklingsprocessens inverkan på testautomatisering: En fallstudie [18] så presenteras teorin i form av 21 krav på utvecklingsprocessen av automatiserad testning. För att ge teoretiskt grund till den första frågeställningen Hur bör ett verktyg för automatisk skriptning och exekvering av regressionstester vara utformat när fysiska komponenter ingår i testobjekt så ska fem utvalda krav ge användas för att ge grund till studien. Urvalen av krav baserade sig på hur väl de kan användas för att förbättra verktygets design och kan ses i Tabell 3 Utvalda krav på lyckad testautomatisering .

Tabell 2 Bergmarks andra krav [18].

2	*Det behöver finnas tydliga specifikationer för hur ett program ska fungera, eftersom den automatiska testningen behöver känna till vad som är ett korrekt beteende.*	Bereza-Jarocinski (2000);

Bergmarks andra krav sett i Tabell 2 ansågs inte vara relevant till verktygets utveckling då en tydlig specifikation för programmets funktion beskrivs i det förbestämda testprotokollet.

Tabell 3 Utvalda krav på lyckad testautomatisering [18].

Nr.	**Krav**	Referens
3	*Det behövs en fungerande spårning av defekter. Rapportering av dessa bör kunna genomföras automatiskt, annars blir det mycket manuellt arbete.*	Bereza-Jarocinski (2000);
4	*Det bör finnas en nedskriven terminologi för testprocessen som kan refereras till för konsekvent användning och förståelse av begrepp och termer*	Persson & Yilmazturk (2004); Parveen et al. (2007);
8	*Testfallen behöver vara tydligt specificerade med förväntat beteende.*	Bereza-Jarocinski (2000);
15	*En analys över risker och fördelar med automatiseringen som ska utföras bör vara gjord.*	Bereza-Jarocinski (2000); Hicks et al (1997); Ng, Murnane, Reed, Grant och Chen (2004);
20	*Man behöver vara medveten om vad det kommer att kosta att automatisera, då kostnaden kan vara mycket hög*	Ng et al. (2004); Taipale, Kasurinen, Karhu, Smolander (2011);

Krav nr.3

Det behövs en fungerande spårning av defekter. Rapportering av dessa bör kunna genomföras automatiskt, annars blir det mycket manuellt arbete.

För att följa detta krav så kommer verktyget automatiskt generera test-utvärderingar efter test. Dessa testutvärderingar kommer lista de fel som uppstod under testning och inkludera en kort beskrivning samt testloggar relaterade till de specifika felen för att en utvecklare ska kunna använda dessa test-utvärderingar för att finna och lösa defekter på testobjektet.

Krav nr.4

Det bör finnas en nedskriven terminologi för testprocessen som kan refereras till för konsekvent användning och förståelse av begrepp och termer

Detta krav kommer delvis att uppfyllas då de funktioner som används i testfallen kommer att presenteras i tabeller för att möjliggöra vidare skapnaden av nya testfall.

Krav nr.8

Testfallen behöver vara tydligt specificerade med förväntat beteende.

Med testspecifikationsspråket TESLA 3.4 så kommer verktyget att försöka följa krav nr.8 genom att analysera och tolka de nuvarande testprotokollens testfall för att möjliggöra test-script och i slutet också förtydliga testfallens indata och utdata (beteende).

Krav nr.15

En analys över risker och fördelar med automatiseringen som ska utföras bör vara gjord.

Analys utförs under rubrik *Förstudie om risker och fördelar* med testautomatiseringen *4.4.*

Krav nr.20

Man behöver vara medveten om vad det kommer att kosta att automatisera, då kostnaden kan vara mycket hög

Ett exempel på tidsinvesteringen för testautomatisering presenteras under 5.2 Frågeställning 2.

3.6 Sensorer

För att kontrollera höjden på bordsbenen under test så behövs sensorer för att mäta längd. Dessa sensorer fungerar som ett gränssnitt emellan testverktygets mjukvara och den riktiga världen. Det finns många typer av sensorer som kan mäta längd med olika för och nackdelar.

Ultraljudssensor

Ultraljudssensor är en distanssensor som mäter längden genom att skicka iväg en kort ljudsignal med en emitter som studsar mot en yta tillbaka till sensorn som mäter tiden som krävs för signalen ska resa igenom luften.

LIDAR

LIDAR är en akronym för ljusdetektering och mätning (Light Detection and Ranging) och är en sensor som använder sig av ultraviolett eller infrarött ljus för att bestämma avstånd. Ljus från till exempel en laser lyser på ett objekt och distansen till ytan kan bestämmas.

Potentiometer och rep

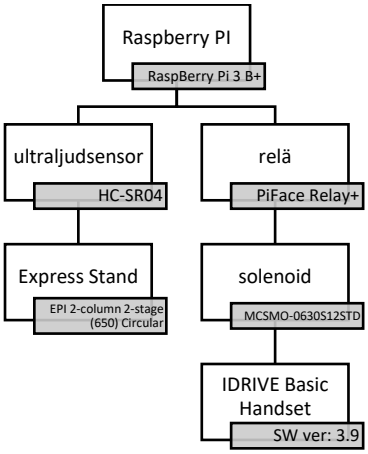
Ett rep och en potentiometer kan användas som en sensor. Om repets ena ände fästs i testobjektet som måste mätas och andra änden i en potentiometer så kan distansen mätas potentiometern kan användas för att omvandla dess resistans till distansen. Professionellt tillverkade linjära potentiometer för distansmätning är mycket dyra då de oftast kan ge en mycket precis distansmätning.

4 Empiri

4.1 Design av verktyg

Design av hårdvara

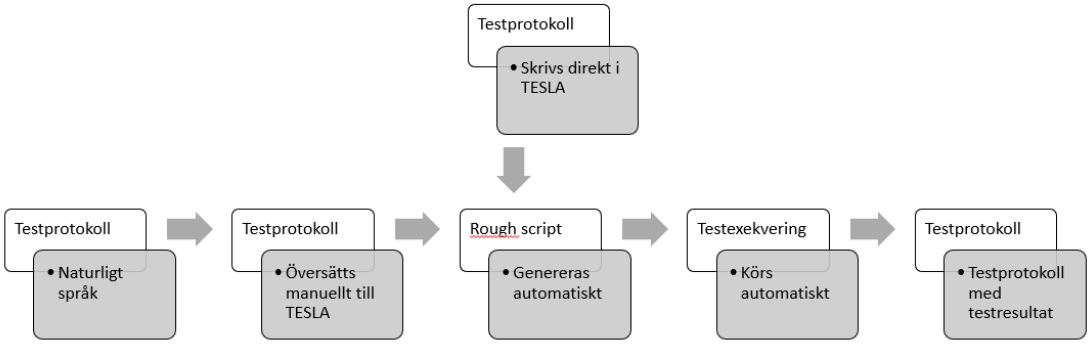
Då testning skulle ske via black-box teststrategin så analyserades testprotokollets indata och utdata. Indata för testobjektet skedde genom fysiska knappar på IDRIVE BASIC HANDSET och utdata bestod av höjden för bordsstativet. För att följa black-box teststrategin så togs hårdvarulösning fram som kunde manipulera och observera testobjektet. Två elektroniska tryckmagneter(solenoider) användes för att hantera den fysiska indata till testobjektet. För att ge spänning åt solenoider så användes ett relä som styrdes genom en Raspberry PI. Raspberry PI valdes delvis då författaren hade tidigare kunskaper om enkortsdatorn. Utdata i form av höjden för bordsbenen behövde mätas så en simpel ultraljudssensor användes.



Val av programmeringsspråk

Ett script-språk behövdes för att automatiskt utföra testexekvering. Python valdes då det ansågs kunna hantera alla delarna i testverktyget. Styrning av relä, hantering av ultraljudssensor, skrivning/hämtning i ett Exceldokument och skript skedde genom Python.

Design av mjukvara



Figur 5 Testprocess-modell för verktyg

För att visa hur verktyget exekverar regressionstestning så framtogs en modell för testprocessen (se Figur 5 Testprocess-modell för verktyg). Processen börjar med ett testprotokoll skrivet direkt i TESLA eller översättning av ett befintligt testprotokoll där testfallen manuellt översätts till TESLA. Krav nr 4 i Bergmarks samanställda krav på lyckad testautomatisering påpekade på att terminologin i testprocessen bör skrivas ner så en mall för de olika kommandon skrevs ner (se Bilaga 1). Testprotokollet används då för att generera ett

”Rough script” som kan ändras vid behov. Skriptet hanterar testexekveringen som sker automatiskt och ett testprotokoll med testresultat genereras efter skriptet kört klart.

4.2 Analys av testfall och applicering av TESLA

I fall då regressionstestning utförs manuellt så utförs testningen enligt ett testprotokoll. Detta testprotokoll presenterar flera testfall som ska utföras och ger beskrivningar till exempel om testfallets syfte, utförande och det förväntade resultatet. Testprotokollet ger en tydlig struktur som kan repeteras presenterar testutförande i ett naturligt språk som kan förstås av både erfarna och oerfarna testare. Ett problem med naturligt språk är att det ej är anpassat för automatisering då det ofta är mångtydigt och typiskt saknar struktur [25]. För att möjliggöra automatisering och ge mer tydliga testfallbeskrivningar så ”översattes” testfallen till TESLA då tydliga testfall följde krav nr 8 i Bergmarks samanställda krav på lyckad testautomatisering.

I inbyggda system så hanterar TESLA:s actionblock indata till testobjektet. Denna indata kan till exempel bestå av in-variabler eller tillstånd. Om vi tittar på ett testfall för testning av bordsben (se Tabell 4) så ansågs indata till testobjektet vara det interface som höjde och sänkte bordsbenen. Interfacet bestod av höj och en sänk-knapparna på handset där actionblocket styrde om knapparna var nedtryckta eller inte nedtryckta. Utdata ansågs vara bordsbenens höjd ifrån marken. Verktyget utvecklades under två iterationer, första iterationens mål var att kunna köra regressionstester automatiskt och andra iterationens syfte var att kunna förbättra verktyget med resultatet ifrån den första iterationen.

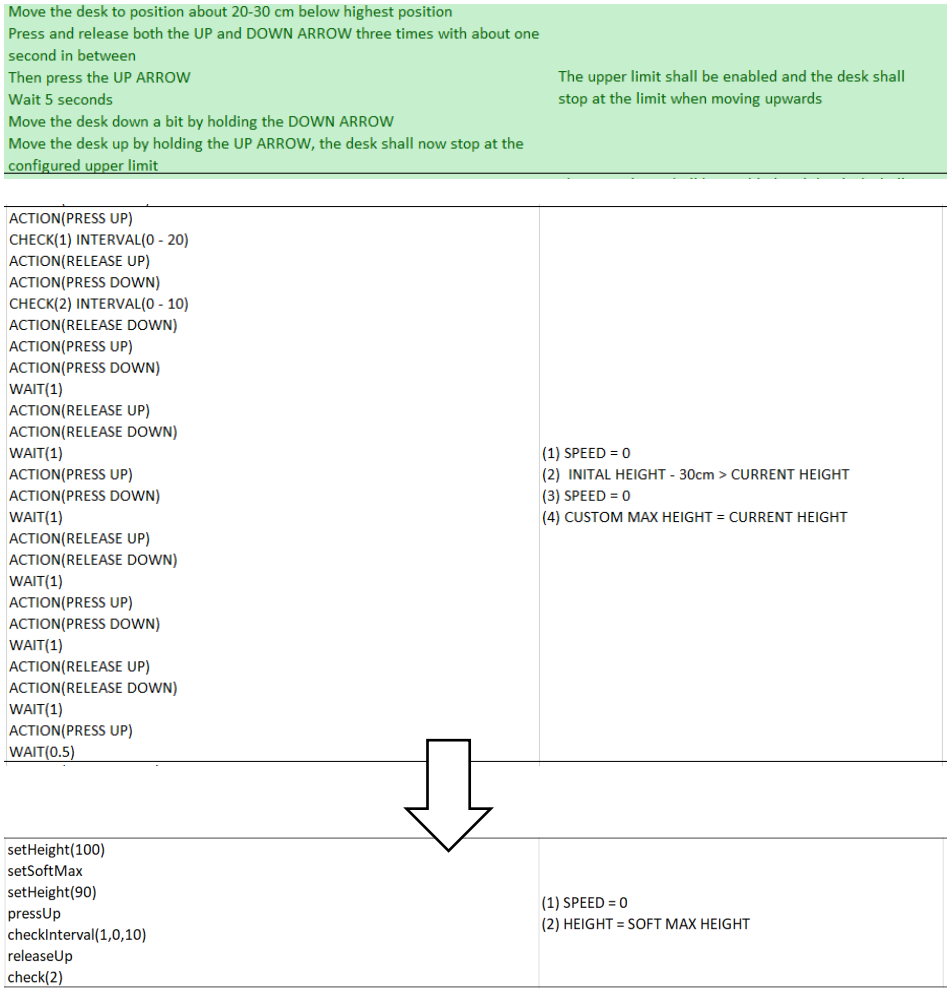
Tabell 4 Exempel på manuellt testfall (grön rad) och ett testfall översatt till TESLA under första designiterationen (vit rad)

Test ID #	Test	Test purpose	Detailed instructions (indata)	Expected result (utdata)	Test result PASS / FAIL / DP FAILED
HB002	Move desk up	Check that the desk is moving up when pressing the Up Arrow/Plus Sign	Press and hold the up arrow	The desk shall move up	PASS
HB002	Move desk up	Check that the desk is moving up when pressing the Up Arrow/Plus Sign	ACTION(PRESS UP) CHECK(1) INTERVAL(1 - 10) ACTION(RELEASE UP)	(1) INITIAL HEIGHT < CURRENT HEIGHT	

I Tabell 4 så syns ett exempel på hur ett testfall ”översattes” under första design-iterationen. De kommandon som beskrivs under input utförs sekventiellt efter varandra. ACTION(pressUp) trycker först på upp-knapp på handset och CHECK(1) INTERVAL(1 - 10) kollar om output nr 1 (INITIAL HEIGHT < CURRENT HEIGHT) är sant under ett intervall, ACTION(releaseUp) släpper upp-knappen därefter. Kontrollen(check) övervakar utdata och godkänner eller underkänner testfallet om predikatet inte uppstår under det bestämda tidsintervallet. Under rubrik TESLA 3.4 så beskrivs de olika typerna av kontroller. När TESLA används i tester av inbyggda system så används processorns klockcyklar för att bestämma tidsintervallet. När checkblock skulle implementeras i verktyget så användes de formler [25] som beskriver kontrollernas funktionalitet. Då formlerna använder sig av en starttid och sluttid för att bestämma tidsintervallet så användes sekunder för att beskriva intervall i testverktyget.

Iteration 2

Under den andra designiterationen så krävdes en förenkling av testfallen. Användning av action och checkblock höll en simpel och förståbar struktur under de simplare testfallen men i mer komplexa testfall som i Figur 6 så blev de mer och mer komplicerade och helheten av testfallen var svårt att uppfatta. Då detta går emot TESLAS grundtanke att förkorta testfallen så behövde något förändras. ”Helper-funktioner implementerades för att ge kortare testfall, kommandon ändrades för till att vara mer lättförståeliga och fler typer av checkblock implementerades för att utöka funktionaliteten för verktyget. Alla kommandon som kan användas för att översättning till TESLA kan ses under Bilaga 1.



Figur 6 Förenkling av TESLA

4.3 Empiriska resultat av experiment

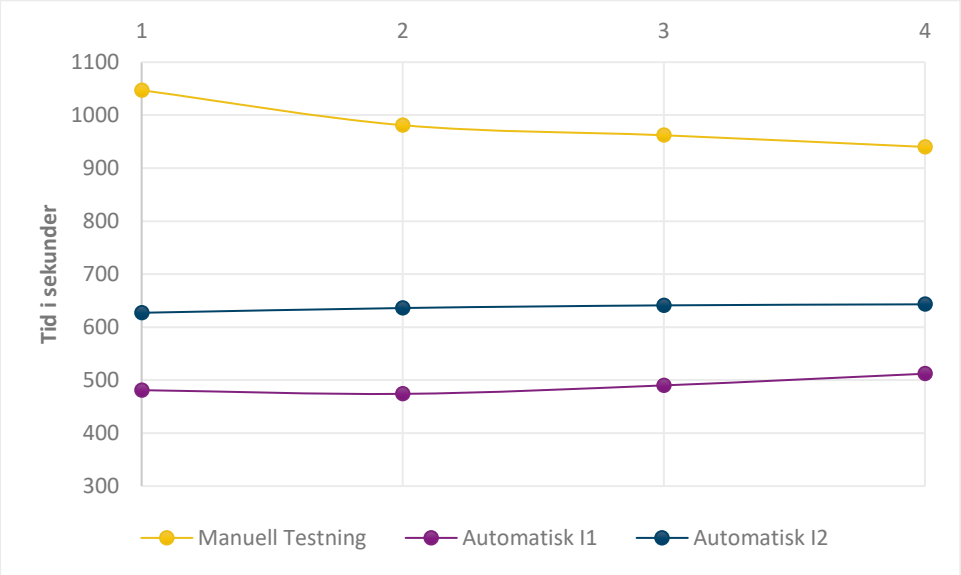
Tiderna sett i Tabell 5 framtogs under 3 typer av experiment. Den första delen av experimenten utfördes genom att utföra ett testprotokoll manuellt. Det manuella experimentets tid mättes med hjälp av ett tidtagarur, tiduret startades och författaren läste testfallen och utfördes de instruktioner som stod beskrivna för hand. Tiduret stoppades då alla testfallen var genomförda. I dessa experimenten så var testfallen skrivna i naturligt språk. Utförande av alla typer av experiment repeterades fyra gånger efter varandra för att ge ett mer pålitligt medelvärde. Då verktyget ansågs kunna utföra testprotokollet automatiskt och få samma resultat som de manuella testerna så ansågs den första iterationen av utveckling vara

klar. Testet exekverades här fyra gånger med samma skript och samma kod för verktyget. Utförandet av testprotokollet tid mättes likt de manuella testerna med tidur. Verktyget omarbetades under den andra iterationen tills den ansågs kunna utföra testprotokollet med samma resultat som de manuella testerna. Exekvering mättes här med pytonmodulen ”time” för att lättare kunna mäta exekveringstiden för testprotokollet. Alla tider presenteras grafiskt i Figur 7 Punktdiagram testtider samt mer detaljerat under Bilaga 2.

Tabell 5 Exekveringstider (minuter) för ett testprotokoll

Test Nr:	*1*	*2*	*3*	*4*	*Medel*	σ
Manuell test	17,5	16,4	16,0	15,7	16min 20s	~40s
Autotest iteration 1	8,0	7,9	8,2	8,5	8min 9s	~14s
Autotest iteration 2	10,5	10,6	10,7	10,7	10min 37s	~6s

Första iterationens testverktyg minskade exekveringstiden för testprotokollet med ca 50% (0,501) gentemot den manuella testningen.



Figur 7 Punktdiagram testtider

Andra iterationers testverktyg minskade exekveringstiden för testprotokollet med 35% gentemot den manuella testningen och därmed ökade den första iterationens exekveringstid med ca 30% (1,303).

Dessa resultat presenteras i Figur 7 och diskussion av resultaten sker under rubriken Frågeställning 2.

4.4 Förstudie om risker och fördelar med testautomatiseringen

En kort analys över de risker och fördelar som kunde uppstå under detta automatiseringsprojekt gjordes under utveckling för att följa krav 15 av Bergmarks samanställda krav på lyckad testautomatisering. Förstudien framtogs genom samtal med ROL:s testare, utvecklare och delvis av författarens egna åsikter och observationer. Förstudien utfördes mest bara för att följa Bergmarks krav och signifikansen av denna analys ansågs vara låg innan testverktygets utveckling.

- Utvecklingstid för testverktyg kan vara svårt att estimera.

De fysiska komponenterna i testobjektet kommer kräva ett mer anpassat testverktyg och tidsestimering med hjälp av liknande testobjekt kan vara svår.

- Testverktyg kan behöva modifieras för olika handset

Då flera typer av handset och bordsbens kombinationer kan användas med varandra så kan detta leda till att testverktyg behöver modifieras för de olika kombinationerna.

- Automatiseringen kan minska tiden som krävs för manuella tester

Då mindre tid spenderas på manuella tester så kan detta ge mer tid över till utvecklare och testare.

- Minska regressionstesternas påverkan av den mänskliga faktorn

Med automatisk regressionstestning så kommer testerna utföras nästan identiskt varje gång och förhoppningsvis minska mänskliga faktorns påverkan på regressionstesterna.

5 Analys

5.1 Frågeställning 1

En artefakt har tagits fram med hjälp av Design Science Research. Artefakten består av ett verktyg som kan utföra automatiska regressionstester på ett höj och sänkbart bordsstativ styrt av ett handset. Ett testprotokoll för manuell regressionstestning fanns innan arbetets start. Ett behov av att automatisera detta testprotokollet ledde till att TESLA användes för att möjliggöra testautomatisering. TESLA valdes då det beskrevs kunna möjliggöra automatisering, översätta normala testfall, ge tydligare testfall och kunde fortfarande representeras likt de gamla testfallen som möjligt. Artefakten visar att användande och anpassandet av TESLA för automatisk skriptning och exekvering av regressionstester var möjliga när fysiska komponenter ingår i ett testobjekt. Användning av TESLA minskade längden av testfallen och minskade mångtydligheten med testfallen skrivna i naturligt språk men försökte ändå hålla en lättförståelig nivå.

Hårdvarulösningen för testverktyget designades för en kortsiktig lösning då verktyget skulle fungera som ett bevis för koncept för automatisering. Användning av tryckmagneter ledde till överhettningar som krävde att testverktyget behövde kylas ner efter tre eller fyra testutföranden för att inte ge opålitliga indata då tryckkraften för solenoiderna minskade. Användning av en ultraljudsensor gav ofta opålitliga resultat på vissa golvytor och i samband med en Raspberry Pi som hanterar sina processer asynkront så gav detta oprecisa höjdvärden. Riskerna med användning av dessa kortsiktiga hårdvarulösningar var kända innan implementation men användes för att få en snabb och kostnadseffektiv lösning. Användning av till exempel pneumatiska cylindrar och en mer precis sensor skulle förmodligen helt eliminerat dessa problem. Hårdvaruproblemen förstörde däremot inte resultaten av de automatiska regressionstesterna då testresultaten kunde i efterhand analyseras för att se om ett misslyckat testfall var ett resultat av hårdvarufel.

Några av Bergmarks samanställda krav på lyckad testautomatisering har använts som vägledning för vad som utgör ett bra automatiseringsprojekt.

Krav nr.3

> *Det behövs en fungerande spårning av defekter. Rapportering av dessa bör kunna genomföras automatiskt, annars blir det mycket manuellt arbete.*

För att följa detta krav så kommer verktyget automatiskt generera test-utvärderingar efter test. Dessa testutvärderingar kommer lista de fel som uppstod under testning och inkludera en kort beskrivning samt testloggar relaterade till de specifika felen för att en utvecklare ska kunna använda dessa test-utvärderingar för att finna och lösa defekter på testobjektet.

Krav nr.4

> *Det bör finnas en nedskriven terminologi för testprocessen som kan refereras till för konsekvent användning och förståelse av begrepp och termer*

Bergmarks fjärde krav motiverade skapandet av tabeller över testverktygets kommando vilket kan användas vid skapandet av nya testfall i TESLA antingen vid översättning av tidigare testprotokoll eller skapande av ett nytt testprotokoll direkt med TESLA.

Krav nr.8

> *Testfallen behöver vara tydligt specificerade med förväntat beteende.*

Krav nummer åtta motiverade fortsatt användningen av TESLA. Användning av TESLA användes inte bara med tanken att kunna möjliggöra automatisering utan även för att ge mer tydligare specificerade testfallens indata och utdata. TESLA säkerställde det förväntade beteendet genom att minska de mångtydiga beskrivningarna i de tidigare testfallen. Detta gynnade regressionstestning då identisk körning av testerna är bra för att hitta oförutväntade fel.

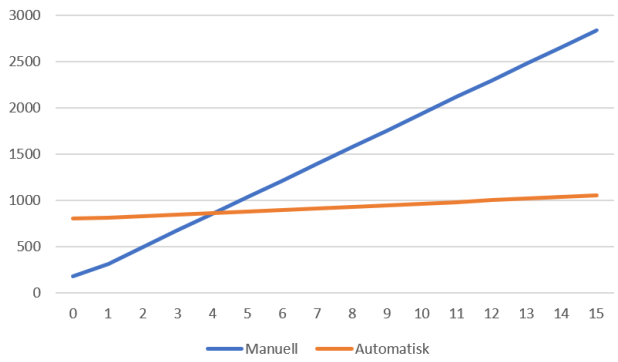
Under rubriken 5.2 så beskrivs hur allvarligt minskningen av testernas exekveringstid var för resultatet av testautomatisering. Arbetets utveckling fokuserade främst på att utveckla

verktyget för att utföra tester med så likt resultat som de manuella regressionstesterna så var tidsexekvering inte en hög prioritet under utveckling. En större fokusering på att minska exekveringstiden hade varit mycket gynnsam för testverktygets resultat då detta är mycket viktigt del i en lyckad testautomatisering.

5.2 Frågeställning 2

Med användning av verktyget så kunde en tidsbesparing på 35% uppnås gentemot manuella exekveringstid. Experiment tidigare i utvecklingen visade en minskning på ca 50% exekveringstiden. Vidareutvecklandet av testverktyget ledde då till en ökad testexekveringstid med 15 procentenheter. Under andra iterationens utveckling av testverktyget så lades mer fokus på att säkra resultatet av testverktyget. En fokus på att minska testfallens påverkan av varandra ledde till att bordet oftare behövde återställas vilket krävde mer tid. Tidsbesparing fick då offras för att ge en högre validitet på testverktygets resultat.
Return on investment är en term som ofta relateras till automatisering. Med en större initial investering så syftar automatiseringen att ge avkastning på denna initiala investering. Om vi tittar på en fallstudie som beskrivs av Graham och Fewster [4] så kan vi se ett jämförande med manuell och automatisk testning. Figur 8 visar att automatiseringen krävde ca 340% större tidsinvestering innan den första testcykeln. Denna investering tjänade in de spenderade timmarna genom att effektivisera testcykeln. Exekveringstiden minskade till $\frac{1}{11}$ av den manuella testtiden och den tid som spenderades på underhåll av tester minskade med $\frac{1}{9}$. Detta ledde till en mycket förminskad testcykel.

Testcykel	*Manuell test tid(h)*	*Automatisk testad(h)*
0	180	800
1	315	812
2	495	829
3	675	846
4	855	863
5	1035	880
6	1215	897
7	1395	914
8	1575	931
9	1755	948
10	1935	965



Figur 8 Tidsbesparing av testautomatisering [14]

Skillnaderna mellan verktyget i denna rapport och fallstudiens verktyg är många. Mjukvarutester som kan minska tidsbesparing flertal gånger med automatisering och ett fokus på automatisk testfallsgenerering som minskar underhåll av tester gör att jämförelse med arbetets testverktyg inte är lämpligt. Fallstudien visar däremot varför en effektiviserad testcykel är nyckeln till att få avkastning på den initiala investering som krävds för automatisering. Tiden som krävs under testcykeln behöver då minskad testexekvering och eller minskat underhåll.

När fysiska komponenter ingår i ett testobjekt så finns risken att exekveringstiden kan begränsas. Automatisering av dessa testobjekt bör noga analyseras innan automatisering för att försäkra sig om hur de fysiska komponenterna kommer påverka testets exekveringstid.

5.3 Frågeställning 3

I verktyget så användes en ultraljudssensor med ca 3mm upplösning. Vid manuella tester så användes ej mätverktyg vid testutförning. All mätning under de manuella testerna utfördes med ögonmått vilket kan variera ifrån person till person men förmodligen så kan det antas att

ögonmåttet inte är lika precist som en ultraljudssensor. En viktigare fråga att ställa än skillnaden i mätnoggrannhet är nog däremot hur mätnoggrannheten faktiskt påverkade testfallens resultat. Ett exempel på ett testfall var att se om bordsbenen kunde höjas eller sänkas. I detta testfall så är förmodligen mätnoggrannhetens påverkan på testfallets resultat mycket lågt. Två tydliga fel som kan uppstå under testfallet är att bordsbenen inte höjs alls eller att bordsbenen sänks istället för att höjas. Båda dessa fall kräver mycket låg mätnoggrannhet. Då testfallen skrevs först för manuellt utförande och senare anpassades för automatisering med en för strikt tolkning så antog förmodligen testaren som skrev testfallen alla testfallet endast kunde utföras med den originella mätnoggrannheten. Om vi istället hade anpassat testfallen mer för en högre mätnoggrannhet så kunde till exempel hastigheten på bordet eller tiden som krävdes för att bordsstativet skulle röra på sig testats samtidigt. En högre mätnoggrannhet bidrar då inte bara till att ett testfall mäts mer noggrant utan att testfallen kan anpassas med den extra precisa noggrannheten för att bli bättre testfall som testar mer.

6 Diskussion och Slutsatser

6.1 Resultat

Genom design science research så har arbetet tagit fram en artefakt som visar att automatisk regressionstestning av testobjekt med fysiska komponenter är möjligt. Automatisering av regressionstestning passade mycket bra för då testerna utförs frekvent och leder till mycket arbete som repeteras. Arbetet har beskrivit TESLA, ett språk för specificering av testfall och hur detta implementerades i testverktyget för att möjliggöra automatisk skriptning och automatisk exekvering av regressionstester. Genom att använda black-boxtestning så analyserades indata och utdata i ett tidigare skrivet testprotokoll. Med analysen så kunde detta testprotokoll ”översättas” till TESLA som möjliggjorde testautomatisering men ändå kunde skapa en testrepresentation som var likt det tidigare testprotokollet. Bergmarks krav för lyckad automatisering användes under utveckling av testverktyget och har gynnat testverktygets resultat.

Genom experiment så har de manuella och automatiska tiderna för utförande av ett testprotokoll mätts. Det manuella testtiderna jämfördes med de automatiska testtiderna och en tidsbesparing på 35% uppnåddes då med det färdiga testverktyget. Tidsbesparing upp till 50% uppnåddes med verktyget men ansågs kunna utföras testerna men inte hålla samma kvalitativa nivå som de manuella testerna. Arbetet har visat med hjälp av en tidigare fallstudie att testautomatisering kräver en större initial tidsinvestering än manuella tester. Syftet med denna initiala investering som krävs att testautomatisering är att effektivisera testcykeln tillräckligt så att den totala testtiden minskar. Det som påverkar testcykeln är testernas exekverings- och tid spenderad på underhåll av tester. Om syftet med automatisering är att få avkastning med automatiseringen så betyder detta att automatiseringen inte kommer vara gynnsam om tidsbesparingen som uppstår vid medgav testautomatisering inte täcker den initiala tidsinvesteringen. Arbetet har visat att riskerna med att automatisera tester med fysiska testobjekt är höga då de fysiska komponenterna kan påverka testets exekveringstid negativt och upphäva syftet med testautomatiseringen.

Testverktyget kunde uppnå en högre mätnoggrannhet med användning av en ultraljudssensor då ögonmåttet ansågs ha en högre upplösning än ögonmåtten som användes i de manuella testerna. Däremot så har en analys av mätnoggrannhetens påverkan på testresultaten visat få skillnader på resultaten av en låg och hög mätnoggrannhet på testerna. Testprotokollet som användes i arbetet var ett befintligt testprotokoll som användes till manuella tester. Testfallen designades då initialt för användning endast i manuella tester med manuell mätnoggrannhet. Om testfallen anpassades för automatisering så kunde t.ex. precisa hastighetsmätningar och fördröjningsmätningar kunna höja kvalitén på testresultatet.

6.2 Implikationer

Testverktygets påverkan på tiden som spenderas på underhåll av tester har inte analyserats i detta arbete. Förmodligen skulle en färdig testrigg för automatiska tester kunna minska tiden för underhåll av tester då mindre tid behöver spenderas för att sätta upp testerna. Sannolikheten är däremot stor att verktygets testfall behöver mer tid att anpassas då buggar och fel med testverktyget behöver fixas och kan då öka hur mycket testerna måste underhållas som måste utföras.
Många av antaganden i detta arbete är att ROI är den största anledningen till testautomatisering. Även om en testautomatisering kan effektivisera testcykeln så att den ger samma totala spenderade tid som manuella tester så kan detta i detta fall avfärdas då avkastningen blir 0%. Verkligheten av testning är inte linjär då intensiteten av antalet tester som krävs varierar mycket baserat på t.ex. utvecklingsstadiet och när släppandet av nya programversioner sker. Under de tidigare utvecklingsfaserna för ett testobjekt så kan funktionaliteterna för mjukvaran ej vara implementerade och behöver kanske mindre testning. En samtidig automatisering med utveckling kan leda till att testare kan förbereda för de mer intensiva testerna som kommer att ske senare. En mer effektiv testcykel ger också mer tid åt testarna att utföra fler tester som kan öka kvaliteten på produkten speciellt då en ny ”release” av mjukvaran släpps.

Rubriken Datainsamling 2.5 beskrev en nollhypotes för experimenten. Nollhypotes för experimenten var att resultaten inte skulle visa någon tidsbesparing med testautomatiseringen och att det manuella utförandet av testprotokollet är snabbare. Experimenten visade att testexekveringstiden minskade då testerna automatiserades och därmed så kan vi förkasta nollhypotes. Ett problem med denna nollhypotes är att experimenten endast omfattar ett litet område av testningen. Experimenten som omfattar utveckling av testverktyget och flera testcyklar så skulle detta förmodligen uppfylla denna nollhypotes.

6.3 Begränsningar

Resultatet av automatiseringens tidsbesparing påverkas självklart av hur mycket manuell testning som utfördes innan. Regressionstestning omfattar ofta mycket tid som krävs till tidskrävande och repetera manuella tester. Arbetet begränsades då ingen empiriska data om den manuella testningen på ROL insamlades och kan då bara göra antaganden för den faktiska data om manuella tester.
En annan stor begränsning med användning av bordsben som testobjekt är att komplexiteten på produkten är relativt låg och har få funktioner som inte kräver mycket tid för att testas.

6.4 Slutsatser och rekommendationer

Användning av TESLA för att översätta testfall till skript för automatiska regressionstester har lyckats även då fysiska komponenter ingår i testobjektet.
Med automatisering av tester så kommer en stor initial tidsinvestering krävas för att automatisera testerna. Med denna initiala investering så syftar de automatiska testerna att kunna effektivisera testexekvering och minska underhållen av tester tillräckligt för att den totala tiden som spenderar på tester minskas. Man behöver vara medveten att automatiseringens risker då kostnaden för testautomatisering kan vara mycket höga. Implementering av testautomatisering utan en analys om dess risker och fördelar kan leda till en alldeles för stor initial tidsinvestering som inte effektiviserar testcykeln tillräckligt för att motivera automatisering av tester. Testautomatisering för tester för testobjekt med fysiska komponenter leder en stor risk att tidsbesparingen med automatiseringen kan begränsas av de fysiska gränserna av komponenterna.
Med hjälp av en utvecklare på ROL så kunde en ungefärlig data på de manuella testtiderna på ROL estimeras. Utvecklaren beskrev att utförandet av testfallet som verktyget hanterade tog cirka 30 minuter för ett genomförande av ett testprotokoll. Om vi antar att mjukvara ändras i genomsnitt en gång per arbetsdag i sex månader (26 veckor) och att fem typer av bordsben som ska testas med ett handset så kan en ekvation skapas för att se hur många utvecklingstimmar som kan läggas på verktyget under perioden för att ändå behålla samma totala testtid som de manuella testerna.

$$tid\ sparad\ per\ automatiskt\ test * antal\ tester = tid\ sparad\ av\ automatisering$$

$$(30\ \text{min} - 10\ \text{min}\ 37s) * (\ 26veckor * 1\ test\ per\ dag * 5bordstyper) \approx 40\ timmar$$

Även med en generös beräkning på tiden besparad av automatiska tester och med ett mål på 0% avkastning så är testautomation svårt att motivera i denna typen av testsituation.

Arbetet tittade också på hur mätnoggrannheten ändrades med användningen av verktyget. En ökad mätnoggrannhet i testfall har en liten påverkan på testfallens resultat om den ökade mätnoggrannheten inte övervägs i design av testfallen.

6.5 Vidare forskning

Ett annat sätt att effektivisera testcykeln var att minska den tiden som krävdes för underhållen av tester. Forskning av automatisk testgenerering visade att detta kunde minska tid för underhåll av tester flera gånger om [4]. Denna typ av automatisering kanske till och med kan vara mer gynnsam för tester för testobjekt med fysiska komponenter då de fysiska komponenterna förmodligen har en mindre påverkan på denna automatisering.

Ingen data om hur mycket testning som egentligen utförs genom manuella tester fanns inom arbetet. En insikt i hur mycket tid som går till manuell testning kunde ge mer insikt till hur viktigt testautomatisering är.

7 Referenser

[1] H. H. Emely, Apache JMeter, Packt Publishing, 2008.

[2] G. Webb och N. Patton, ”Quality and Cost – It’s Not Either/Or: Making the Case With Cost of Quality,” *CROSSTALK,* vol. 21, nr 11, pp. 27-28, 2008.

[3] D. Graham och M. Fewster, ”Reflections on the Case Studies,” i *Experiences of Test Automation: Case Studies of Software Test Automation*, New York, Addison-Wesley Professional, 2012, pp. 1-16.

[4] D. Graham och M. Fewster, ”Model-Based Test-Case Generation in ESA Projects,” i *Experiences of Test Automation: Case Studies of Software Test Automation*, Addison-Wesley Professional, 2012, pp. 155-175.

[5] S. Dekker, ”The New View of Human Error,” i *The Field Guide to Understanding Human Error, 2nd Edition*, Lund, Ashgate Publishing Company, 2006, p. 2.

[6] D. Stockdale, ”Let’s Talk About Automated Regression Testing,” Testlio, 22 September 2017. [Online]. Available: https://testlio.com/blog/regression-testing-automated/. [Använd 27 Juni 2018].

[7] S. Woody, ”The One-Hour Regression Test,” *better software,* vol. 2008, nr 08, pp. 24-28, 2008.

[8] M. Thoss, K. Beckmann, R. Kroeger, M. Muenchhof och C. Mellert, ”A Framework-Based Approach for Automated Testing of CNC Firmware,” i *ISSTA '14 International Symposium on Software Testing and Analysis*, San Jose, 2014.

[9] B. Venners, ”The Making of Python,” Artima, 13 January 2003. [Online]. Available: http://www.artima.com/intv/pythonP.html. [Använd 19 6 2018].

[10] D. . Sheppard, ”Beginner's Introduction to Perl,” , . [Online]. Available: http://www.perl.com/pub/2000/10/begperl1.html. [Använd 19 6 2018].

[11] G. Chapelle, ”A practical guide to linux commands, editors, and shell-programming, third edition by Mark G. Sobell,” *ACM Sigsoft Software Engineering Notes,* vol. 38, nr 4, p. 255, 2013.

[12] E. community, ”Github,” Ecma International, 18 Juni 2018. [Online]. Available: https://tc39.github.io/ecma262/. [Använd 19 Juni 2018].

[13] J. K. Ousterhout, ”Scripting: HigherLevel Programming for the 21st Century,” *Computer,* vol. 31, nr 3, pp. 23-30, 1998.

[14] I. Muneer, ”Systematic Review on Automated Testing Types, Effort and ROI,” Linköpings universitet, Linköping, 2014.

[15] J. Z. Gao, H.-S. J. Tsao och Y. Wu, ”Testing and Quality Assurance for Component-Based Software,” i *Testing and Quality Assurance for Component-Based Software*, Norwood, Artech House, 2003, pp. 119-126.

[16] V. R. Basil och B. T. Perricone, ”SOFTWARE ERRORS AND COMPLEXITY: AN IMPERICAL INVESTIGATION,” *Communications of the ACM ,* vol. 27, nr 1, pp. 42-52, 1984.

[17] I. Muneer, ”Systematic Review on Automated Testing Types, Effort and ROI,” Linköpings universitet, Linköping, 2014.

[18] R. Bergmark, ”Utvecklingsprocessens inverkan på testautomatisering: En fallstudie,” 2012. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:his:diva-6102. [Använd 4 Maj 2018].

[19] J. Venable och J. Iivari, ”Action research and design science research - Seemingly similar but decisively dissimilar,” i *Conference: Conference: 17th European Conference on Information Systems, ECIS 2009*, Verona, 2009.

[20] P. Johannesson och E. Perjons, ”A Method Framework for Design Science Research,” i *A Method Framework for Design Science Research*, Kistad, Springer International Publishing, 2014, pp. 75-88.

[21] P. Blomkvist och A. Hallin, ”FÖRHÅLLANDET MELLAN TEORI OCH EMPIRI: INDUKTION, DEDUKTION, ABDUKTION,” i *Metoder för teknologer*, Lund, Studielitteratur AB, 2015, pp. 45-46.

[22] D. E, R. J och P. J, ” Automated Software Testing: introduction, management and performance,” Addison-Wesley, 1999.

[23] T. Crowder, ”StackOverflow,” Stack Exchange, Inc., 22 Juni 13. [Online]. Available: https://stackoverflow.com/a/17253557/5996975. [Använd 5 September 2018].

[24] Google, ”Google Developers,” Google, 17 november 2011. [Online]. Available: https://developers.google.com/v8/. [Använd 5 september 2018].

[25] M. Wahler, E. Ferranti, R. Steiger, R. Jain och K. Nagy, ”CAST: Automating Software Tests for Embedded Systems,” i *CAST: Automating Software Tests for Embedded Systems*, Montreal, 2012.

[26] KLS Sharma, ”Why automation?,” i *Overview on Industrial Process Automation*, Bangalore, Elsevier, 2011, pp. 1-14.

8 Tabell över figurer

Bilagor

Bilaga 1 Kommando för att översätta TESLA

Iteration1

ACTION(command)	CHECK(output) INTERVAL(t1 - t2)
PRESS UP	x = y
RELEASE UP	x > y
PRESS DOWN	x < y
RELEASE DOWN	

Iteration2

ACTION	CHECK	OUTPUT
pressUp	checkInterval(output nr,start,end)	INITIAL HEIGHT < CURRENT HEIGHT
pressDown	checkNotInInterval(output nr,start,end)	INITIAL HEIGHT > CURRENT HEIGHT
releaseUp	check(output nr)	HEIGHT = HARD MAX HEIGHT
releaseDown	checkNot(output nr)	HEIGHT = HARD MIN HEIGHT
setHeight(cm)		SPEED = NORMAL SPEED
reset		SPEED = 0
setSoftMax		HEIGHT > SOFT MAX HEIGHT
setSoftMin		HEIGHT < SOFT MIN HEIGHT
		HEIGHT = SOFT MAX HEIGHT
		HEIGHT = SOFT MIN HEIGHT

Bilaga 2 Detaljerade exekveringstider

Test Nr:	*1*	*2*	*3*	*4*	*Medel*	σ
Manuell Testning	17min 27s	16min 21s	16min 2s	15min 40s	16min 20s	~40s
Automatisk I1	8min 1s	7min 54s	8min 10s	8min 32s	8min 9s	~14s
Automatisk I2	10min 27s	10min 36s	10min 41s	10min 43s	10min 37s	~6s