CHALMERS

Automatiska Styrsystemtester

Examensarbete inom högskoleingenjörsprogrammet Mekatronik

OLOF HAGVALL

ELIAS ÅBERG

INSTITUTIONEN FÖR ELEKTROTEKNIK

CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2025
www.chalmers.se

EXAMENSARBETE INOM HÖGSKOLEINGENJÖRSPROGRAMMET
MEKATRONIK

Automatiska styrsystemtester

OLOF HAGVALL

ELIAS ÅBERG

CHALMERS

Institutionen för elektroteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2025

Automatiska styrsystemtester
OLOF HAGVALL
ELIAS ÅBERG

Handledare: Simon Grimm & Elias Isenstierna, Granitor Systems AB

Examinator: Veronica Olesen, Institutionen för elektroteknik

Institutionen för elektroteknik
Chalmers tekniska högskola
SE-412 96 Göteborg
Sverige
Telephone + 46 (0)31-772 1000

FÖRORD

Detta examensarbete har genomförts på Chalmers tekniska högskola i samarbete med Granitor Systems i Mölndal. Under arbetets gång har vi fått ovärderligt stöd och vägledning av flera personer.

Vi vill först och främst rikta ett varmt tack till våra handledare på Granitor Systems, Elias Isenstierna och Simon Grimm, som först presenterade idén om examensarbetet för oss. De har under ett halvår ihärdigt besvarat våra frågor och stöttat oss med sin kunskap och tid. Vi vill också tacka övrig personal på avdelning 119 Industry och Infra, som varit inkluderande och hjälpsamma från första dagen.

Ett särskilt tack riktas också till vår examinator Veronica Olesen, som hjälpt oss med insikter och konstruktiv återkoppling i rapportarbetet.

Vi är mycket tacksamma för stödet vi fått och hoppas att rapporten speglar det arbete och den omtanke som lagts ned av alla inblandade.

SAMMANFATTNING

Denna rapport undersöker möjligheten att automatisera testprocessen för styrsystem inom industriell automation, med fokus på Programmable Logic Controllers (PLC) och Supervisory Control and Data Acquisition (SCADA). Manuell testning av dessa system är tidskrävande och beroende av leverantörsspecifika plattformar, vilket skapar utmaningar vid utveckling och ändringar av styrsystem. Manuell testning kan också vara felbenägen eftersom den utförs av människor. Detta medför att det finns en risk att teststeg förbises, att instruktioner misstolkas, eller att resultat bedöms subjektivt. Syftet med examensarbetet är att utveckla ett plattformsoberoende testverktyg baserat på Open Platform Communications Unified Architecture (OPC UA), som automatiserar testning av PLC-logik och SCADA-funktioner för att öka effektivitet, minska mänskliga felkällor och sänka kostnader. Verktyget implementeras i Python, använder Microsoft Excel för hantering av testdata och testas mot PLC-miljöer. Genom att simulera driftscenarier och jämföra tidsåtgång och kvalitet med manuell testning utvärderas verktygets prestanda. Rapporten besvarar frågeställningar om verktygets funktionalitet, prestanda och potential att spara resurser. Resultaten visar att ett automatiserat testverktyg kan vara ett användbart hjälpmedel i testprocesser, särskilt vid repetitiva tester med stor mängd styrsignaler eller logik.

ABSTRACT

This report investigates the possibility of automating the testing process for control systems in industrial automation, with a focus on Programmable Logic Controllers (PLC) and Supervisory Control and Data Acquisition (SCADA). Manual testing of these systems is time-consuming and dependent on vendor-specific platforms which pose challenges during the development and modification of control systems. Manual testing is also susceptible to errors, as it relies on human execution. This introduces the risk of test steps being omitted, instructions being misinterpreted, or results being assessed subjectively. The aim of this thesis is to develop a platform-independent testing tool based on Open Platform Communications Unified Architecture (OPC UA), which automates the testing of PLC logic and SCADA functions to improve efficiency, reduce human error, and lower costs. The tool is implemented in Python, uses Microsoft Excel for managing test data, and is tested in both physical and virtual PLC environments. The performance of the tool is evaluated by simulating operational scenarios and comparing its time efficiency and quality against manual testing. The report addresses questions regarding the tool's functionality, performance, and potential for resource savings. The results indicate that an automatic testing tool can be a useful aid in testing processes, especially for repetitive tests involving large numbers of control signals or control logic.

INNEHÅLLSFÖRTECKNING

TERMINOLOGI

OPC – Open Platform Communications,

OPC UA – OPC Unified Architecture,

PLC – Programmable Logic Controller, dator ämnad för styrning av industriella processer.

SCADA – Supervisory Control And Data Acquisition, ett överordnat system som styr och övervakar industriella processer.

HMI – Human-machine interface

IIoT – Industrial Internet of Things

SUT – System under testning

FAT – Factory Acceptance Test

1 INLEDNING

Denna rapport undersöker möjligheten att automatisera testprocesser vid utveckling av styrsystem inom industriell automation.

1.1 Bakgrund

Inom industriella styrsystem används Programmable Logic Controllers (PLC) och Supervisory Control and Data Acquisition (SCADA) för att styra och övervaka processer. En ofta återkommande utmaning när dessa styrsystem utvecklas är att säkerställa en korrekt funktion genom testning. I dagsläget utförs testning av industriella styrsystem manuellt, eftersom det saknas etablerade verktyg för att automatisera testprocessen.

Granitor Systems är systemintegratör och arbetar med utveckling av styrsystem åt olika kunder inom flera industrier i både Sverige och utomlands. Funktionen av de system som utvecklas måste testas för att säkerställa att önskad funktion uppnås och att inga oönskade fel kvarstår vid den slutgiltiga leveransen. Testningen genomförs manuellt och är tidskrävande då systemen som ska testas kan vara komplexa. På grund av denna stora tidsåtgång uppstår ett dilemma om ändringar eventuellt behöver göras i efterhand. Ska utvecklare i så fall behöva spendera stora mängder tid på att testa av hela systemet igen, eller ska enbart vissa delar av styrsystemet testas med förhoppning om att resterande systemfunktion förblir oförändrad? Manuell testning riskerar dessutom att vara felbenägen av faktorer som mänskligt slarv eller trötthet vid stora mängder testade signaler. Utifrån dessa anledningar efterfrågas ett plattformsoberoende verktyg som kan automatisera testprocessen och därmed öka både effektivitet och testernas tillförlitlighet.

Inom mjukvaruutveckling är automatiska tester ett väletablerat koncept med många olika verktyg tillgängliga och har bevisats ge ökad effektivitet och minskade kostnader [1]. Dessutom kan automatiska testverktyg genomföra komplexa testfall med hög tillförlitlighet och på så sätt minska risken för mänskliga fel.

1.2 Syfte

Detta examensarbete syftar till att ta fram ett plattformsoberoende testverktyg som möjliggör automatisk testning av PLC-logik och SCADA-funktioner med hjälp av OPC UA. Målet med verktyget är att automatisera testprocessen, förbättra effektivitet och minska risk för mänskliga fel. Dessutom ämnar arbetet finna verktyg och metoder som kan leda till lägre kostnader, högre utvecklingshastighet och högre säkerhet vid utveckling och implementering av styrsystem. Lösningen utvärderas genom att simulera olika driftscenarier, att testa dessa med testverktyget, och genom att mäta tidsåtgång och kvalitet jämföra med motsvarande manuella testmetoder.

1.3 Avgränsningar

Lösningen utvecklas för PLC- och SCADA-system som stödjer OPC UA. PLC- och SCADA-plattformar som inte har stöd för OPC UA kommer inte att inkluderas i testningen. Styrsystemtestningen kommer att vara inriktad mot grundläggande styrsignaler i styrsystem och innefattar därför inte alla specialfunktioner som kan finnas i specifika PLC- eller SCADA-miljöer. Utveckling sker uteslutande i programmeringsspråket Python, och testning

utförs mot en PLC. För hantering av testdata används Microsoft Excel, som fungerar som lagringsmedium för testfall och resultat.

1.4 Precisering av frågeställning

De frågeställningar som detta examensarbete ämnar besvara sammanfattas enligt följande:

1. Är det möjligt att utveckla ett plattformsoberoende verktyg som genomför automatiserade styrsystemtester med hjälp av OPC UA?
2. Kan verktyget användas för att spara tid vid styrsystemstester?

2 TEKNISK BAKGRUND

I detta avsnitt beskrivs centrala teoretiska begrepp och koncept som behövs för genomförandet av studien.

2.1 Server och klient

En server är en dator eller ett program som är utformad för att hantera data eller erbjuda tjänster och resurser till andra datorer över ett nätverk [2]. En server kan exempelvis vara en filserver för att lagra och hantera filer eller en databasserver. En klient är en dator som använder de tjänster som en server erbjuder [2].

Inom industriella styrsystem kan en server exempelvis vara en PLC som hanterar en databas innehållande alla PLC-taggar (i denna rapport refererar ordet *tagg* till en unik identitet kopplad till en specifik variabel i ett system, exempelvis en sensor eller aktuator). Klienten kan vara ett human-machine interface (HMI) eller ett SCADA-system som hämtar information om taggarna från PLC och presenterar taggarnas värden för användaren. Klienten kan i detta fall även användas för att skicka information till servern.

2.2 Industriella kommunikationsprotokoll

Ett industriellt kommunikationsprotokoll är en uppsättning standardiserade regler och format för informationsöverföring som styr hur mjukvara skickar och tar emot data i en industriell miljö [3]. Dessa protokoll är särskilt utvecklade att sköta kommunikation mellan enheter i industriella automationsprocesser och används vanligtvis i fabriker eller inom infrastruktur, vilket medför att robusthet och tillförlitlighet är centralt för dessa protokoll. Exempel på sådana protokoll inkluderar Modbus, PROFINET och OPC UA.

2.3 OPC och OPC UA

Open Platform Communications (OPC) är en standard för kommunikation inom industriell automation som är utformad för att överföra data mellan server och klient samt mellan server och server [4]. Målet med OPC-standarden är att abstrahera PLC-specifika kommunikations-protokoll till ett standardiserat gränssnitt. Detta gör det möjligt att kommunicera med PLC eller styrenheter av olika fabrikat genom att OPC-applikationen agerar mellanhand och omvandlar generella OPC läs- och skrivbegäran till enhetsspecifika begäran.

Till en början var OPC-standarden begränsad till Windows operativsystem. Dessa refereras vanligtvis till som OPC Classic. OPC Unified Architecture (OPC UA) är en plattforms-oberoende standard som bygger vidare på OPC Classic som tillåter kommunikation med en mängd olika enhetstyper [5].

2.3.1 Nodes

En node eller nod, är ett grundläggande begrepp i OPC UA-protokollet och en OPC UA-server kan innehålla tusentals noder. Varje nod representerar någon form av information. Vad informationen i noden betyder bestäms av nodens typ. En nod kan vara av typ variable, dataType, object, med mera. Detta innebär att all realtidsdata så som PLC-taggar, larm, med mera representeras av noder på OPC UA-servern.

2.3.2 Namespace

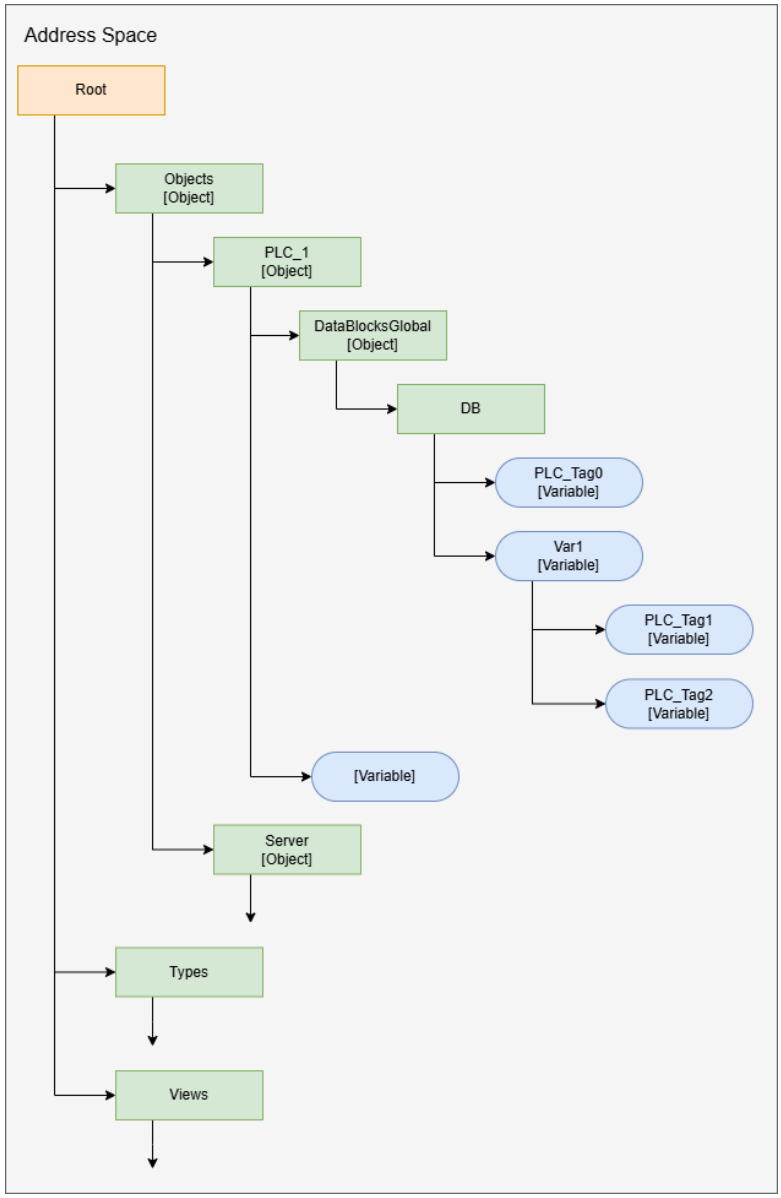
Ett namespace kan beskrivas som en behållare för noder. På en OPC UA server finns det oftast flera olika namespace deklarerade och varje nod tillhör ett av dessa. Detta betyder att man måste titta i rätt namespace för att hitta en specifik nod.

Varje namespace har ett eget indexnummer, ett så kallat namespace-index. I specifikationen för OPC UA är det bestämt att namespace-index 0 och 1 är reserverade för OPC-specifika noder, alltså noder som hör till protokollet [6]. Därutöver kan tillverkare och användare själva definiera nya namespace. Till exempel använder Siemens 1500 PLC-system sig av namespace-index 3.

2.3.3 Address Space och nodeId

Address Space är en strukturerad samling av alla de noder som OPC UA-servern gör tillgängliga för klienten. Den kan likställas med en databas innehållande all data som en klient har tillgång till på servern.

Addressutrymmet i OPC UA är uppbyggt enligt en trädstruktur. Figur 2.1 visar ett exempel på hur denna trädstruktur kan se ut.



Figur 2.1: Exempel på uppbyggnad av adressutrymme på OPC UA-server.

På varje OPC UA server finns alltid startnoden Root på nivå 0, samt noderna Objects, Types och Views på nivå 1, under Root, se Figur 2.1 ovan.

Varje nod har ett unikt identifieringsnamn kallat nodeId. En nods nodeId består av index till det namespace som noden tillhör samt nodens identifier [7]. Om noden inte tillhör en annan nod är dess identifier samma som nodens namn [6]. Om noden tillhör en annan nod kommer dess identifier att bestå av identifier till överordnad nod, en ”.” och sedan namnet på sig själv. Från Figur 2.1 kan det exempelvis se ut enligt följande: DB.Var1.PLC_Tag1. I detta exempel tillhör PLC_Tag1 Var1 som i sin tur tillhör DB, men DB tillhör inte DataBlocksGlobal. Vilka noder som tillhör varandra beror av fabrikatet på den enhet som OPC UA-servern körs på.

2.4 Siemens TIA Portal

Totally Integrated Automation Portal (TIA Portal) är en utvecklingsmiljö utvecklad av Siemens. TIA Portal integrerar flera av Siemens tidigare verktyg till ett och samma program. TIA-portal används för programmering och konfigurering av industriella styrsystem och produkter från Siemens, exempelvis PLC, HMI och frekvensomvandlare [8].

2.5 KepServerEX

KepServerEX är en programvara som kan likställas med en översättare. Den används vanligtvis för att koppla samman olika enhetsspecifika kommunikationsprotokoll genom ett gemensamt gränssnitt, OPC, till en och samma server [9], [10]. KEPServerEX är en användbar lösning för att standardisera kommunikation inom industriella miljöer, vilket underlättar integration mellan olika system.

2.6 Python

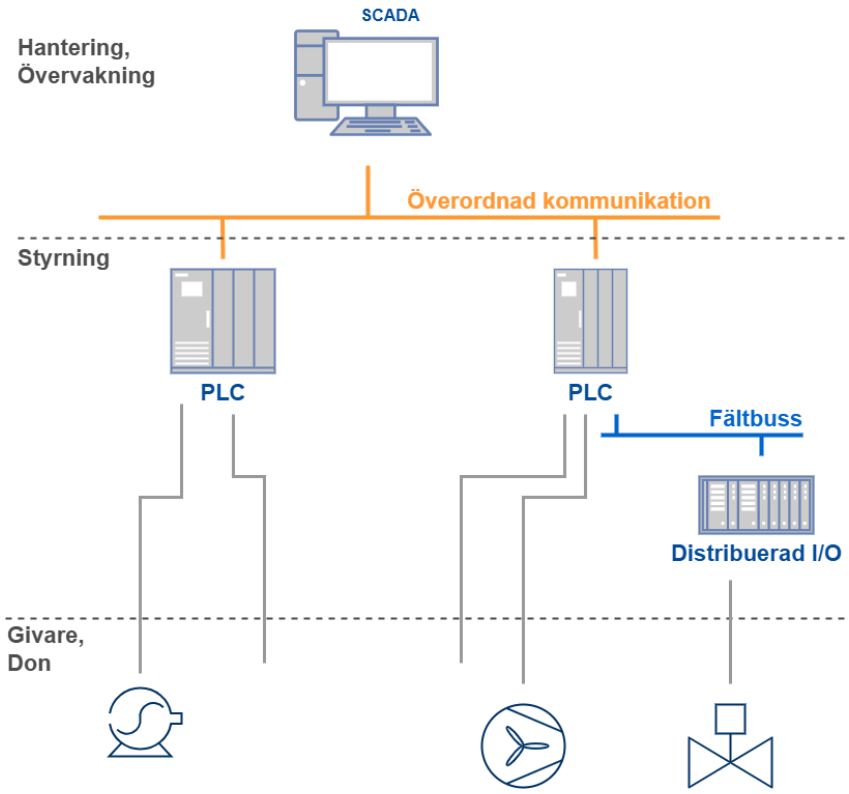
Python är ett högnivåprogrammeringsspråk som introducerades 1991. Språket har stor spridning med många användare vilket med tiden lett till ett stort antal tillgängliga externa bibliotek. Detta har möjliggjort språkets breda användning inom områden som vetenskaplig beräkning, webbutveckling, maskininlärning och automation [11]. Python har via sin omfattande användargemenskap också goda möjligheter till integration med både extern hårdvara och mjukvara, vilket gör det lämpligt för industriella lösningar.

2.7 SCADA

Supervisory Control And Data Acquisition (SCADA) är en mjukvaruarkitektur som används inom industriella styrsystem för att övervaka och fjärrstyra processer via HMI i realtid [12]. SCADA används inom flera industriella sektorer och även inom infrastruktur där de kan övervaka samhällsviktiga funktioner som energisystem, vatten- och avloppssystem eller transportinfra-struktur. Till skillnad från HMI är SCADA ett mer övergripande system som samlar in data från många olika enheter. Där ett HMI huvudsakligen används lokalt vid en enskild maskin, så kan SCADA sträcka sig över en hel anläggning och användas för att övervaka och styra komponenter i olika processer. På senare tid och med fortsatt tillväxt av the Industrial Internet of Things (IIoT) har potentialen för industriell digitalisering och kapaciteten för SCADA-system ökat markant [13].

2.8 Uppbyggnad av styrsystem

Uppbyggnaden av styr- och övervakningssystem för större anläggningar kan ofta innehålla flera nivåer av styrenheter som alla kommunicerar och delar data med varandra. Dessa styrenheter kan vara allt från överordnade SCADA-system, PLC eller distribuerade in- och utgångsenheter. Figur 2.2 nedan visar hur nätverkstopologin för ett industriellt styrsystem skulle kunna se ut.



Figur 2.2: Exempel på kommunikationstopologi hos ett komplett styrsystem.

Längst upp i hierarkin finns SCADA-system som övervakar hela processen. Under SCADA finns PLC-system som styr delar av processen. Inom dessa styrsystem kan det ofta finnas enheter av olika fabrikat. För att kommunikationen då ska fungera smidigt krävs att det används kommunikationsprotokoll som har brett stöd från flera olika fabrikat.

2.9 Testning av mjukvara och styrsystem

Testning av mjukvara är en grundläggande del av programutveckling, med syftet att säkerställa att ett program beter sig som förväntat. Ett testfall består i sin enklaste form av två huvudkomponenter: en beskrivning av den inputdata som används för att initiera testet, samt en specificering av den förväntade outputen som systemet ska generera som svar på denna input [14]. När ett testfall körs skickas inputdata till systemet under testning (SUT) och den faktiska outputen jämförs sedan med det förväntade resultatet. Beroende på om dessa stämmer överens avgörs om testfallet har lyckats eller misslyckats.

Vid denna typ av testning betraktas SUT ofta som en så kallad svart låda (black box), vilket innebär att man inte tar hänsyn till systemets inre funktion eller uppbyggnad. Fokus ligger i stället endast på hur systemet reagerar på en given insignal, och om dess utsignaler motsvarar de som förväntas.

2.9.1 Testorakel

Ett testorakel (från engelska ”Test Oracle”) är inom mjukvaru-världen det begrepp som används för att beskriva den process som avgör om ett testutfall är korrekt eller inte [15]. Oraklet är alltså det objekt som besitter informationen om de korrekta utsignalerna för ett testfall. Det så kallade orakelproblemet, det vill säga problematiken i hur en testprocess ska veta vilket som är det korrekta testutfallet, har historiskt sett varit ett problem vid försök till testautomatisering, då det i slutändan ofta kräver att en människa bekräftar vad som är det korrekta. Detta är en monoton, tidskrävande och ibland även felbenägen process som innebär en signifikant flaskhals för åstadkommandet av total testautomation [16], [15].

2.9.2 Olika typer av testning

Inom mjukvarutestning förekommer en mängd olika testmetoder, varav flera är etablerade och används som praxis i de olika faserna av en utvecklingscykel [17]. Varje testmetod har ett specifikt syfte och inriktar sig på olika aspekter av systemet, vilket gör det möjligt att systematiskt verifiera och validera såväl enskilda komponenter som hela lösningens funktionalitet och prestanda. Metoderna är alltså inte alternativ till varandra utan kompletterar snarare varandra för att få så stor täckning av testningen som möjligt. Nedan följer beskrivningar av några vanligt förekommande testmetoder som brukar användas under olika faser av mjukvaruutveckling.

Enhetstestning

Enhetstestning (från engelska “Unit testing”) är en metod att testa hård-och mjukvara genom att varje individuell komponent eller komponentgrupp testas isolerad [18]. Det är en metod som används för att säkerställa att varje enskild del av systemet uppför sig som förväntat. Därför är enhetstestning ofta särskilt användbar i utvecklingsfasen av ett projekt och kan på detta vis påträffa buggar tidigt under utvecklingsprocessen och förbättra projektets kvalitet och minska dess felbenägenhet [19]. En viktig aspekt att vara medveten om är att enhetstestning är en tid- och arbetskrävande process, samt att det kan vara svårt att isolera komponenter i komplexa system [20].

Integrationstestning

Integrationstestning används för att testa att olika delar av ett system fungerar tillsammans och för att hitta fel där olika delar av systemet kolliderar och ger konfliktfel [21]. På detta vis kan utvecklare identifiera problem som uppstår mellan olika delar av mjukvaran, samt utvärdera effektivitet och samspel mellan olika programvarudelar. Integrationstestning används vanligtvis stegvis efter att nya komponenter integreras i ett system och sker vanligtvis efter enhetstestning när systemets enskilda delar är färdigutvecklade och börjar kombineras [22].

Systemtestning

Systemtestning är den testning som görs innan en produkt kan levereras för acceptanstestning som sker tillsammans med beställaren [23]. Hela systemet testas för att säkerställa funktion och att det uppfyller de kriterier som bestämts. Testfallen som görs är i detta stadie omfattande och med målet att täcka så många som möjligt av systemets olika tillstånd för att hitta buggar och korrigera dessa [24].

Acceptanstestning

Acceptanstestning är en kritisk och avslutande fas inom utvecklingscykeln där utvecklare och beställare tillsammans utvärderar produkten för att försäkra att den uppfyller de utsatta kriterier och funktionella krav som utvecklare och beställare tillsammans kommit överens om [25]. Vid industriella applikationer genomförs vanligtvis acceptanstestning inom det som kallas för *Factory Acceptance Test (FAT).* Under denna testning avgörs huruvida produkten faktiskt är redo att levereras och driftsättas i sin tänkta, operativa miljö.

Regressionstestning

När en produkt väl är levererad och driftsatt övergår utvecklingsfasen till en underhållsfas. Om en uppdatering då görs i ett program finns det en risk att detta introducerar nya, oavsiktliga fel som kan störa funktion och i värsta fall utgöra en säkerhetsrisk [26]. För att minimera denna risk genomförs regressionstestning då förändringar har införts i programmet, vilket i praktiken innebär att tidigare genomförda systemtester måste genomföras igen [26]. Detta kan vara en resurskrävande process för utvecklare som behöver göra omfattande och tidskrävande tester, även vid små justeringar av programvara.

2.9.3 Automatiserad testning

Automatiserad testning är ett begrepp inom mjukvaruutveckling där ett system testas med hjälp av särskild programvara som automatiskt styr utförandet av testfall samt jämför resultat med förväntade utfall [27]. Genom att ersätta manuella arbetsmoment med automatiserade skript kan betydande delar av testprocessen effektiviseras, vilket i sin tur minskar behov av manuell testning [1]. Detta är särskilt fördelaktigt vid repetitiva testförfaranden, där samma scenarier behöver köras upprepade gånger, till exempel vid testning av ett stort antal styrsignaler och tagg-avläsningar. I sådana fall kan automatiserad testning vara särskilt fördelaktig då skalbarheten är god för denna typ av testning. Och ur aspekten av repetitiva och återkommande tester kan automatiserad testning med sitt mer konsekventa genomförande och sin höga exekveringshastighet vara mer lämplig än dess manuella motsvarighet [1].

3 METOD

Detta kapitel beskriver den grundläggande metodik och de verktyg som används för att genomföra studien. För att besvara projektets frågeställning utvecklas ett testverktyg som genomför automatiska styrsystemstester. Programutveckling görs i Python och OPC UA används för kommunikation mot systemets styrenheter. Under programutvecklingen används en Siemens S7-1500 PLC samt KepserverEX som OPC UA-server. Kommunikation mellan enheter och servrar sker på ett lokalt nätverk. Excel används för inmatning av olika testfall samt för avläsning av testresultat.

Testverktyget appliceras på en vägbom inom ett styrsystem som är del av ett större infrastrukturprojekt, där dokumentation för ett redan existerande FAT finns. De tester som genomfördes manuellt på bommen under FAT genomförs med det automatiska testverktyget. Resultat från testningen utvärderas för att besvara frågeställningar.

3.1 Kravspecifikation

En kravlista upprättas för att ha som referenspunkt vid utvecklingen. Då projektet i detta stadie inte avser någon marknadsfärdig, eller kommersiell produkt, utan snarare används som en prototyp för att besvara frågeställningar, är de hårda krav som upprättas ganska få. Mycket utrymme ges till förändring efter hand om nya möjligheter upptäcks. Tabell 3.1 innehåller en sammanställning av de huvsakliga krav som finns, därefter följer en mer detaljerad beskrivning av varje krav.

Tabell 3.1: Lista med krav på testverktyget.

KRAV #	BESKRIVNING
1	Kommunikation sker med OPC UA
2	Verktyget är plattformsoberoende
3	Verktyget kan testa styrsystem med minst två styrenheter
4	Vid fel fås information om vilken specifik tagg som inte klarat test
5	Verktyget ska som minimum kunna testa signaler av datatypen BOOL och INT

1. Verktyget ska använda OPC UA som kommunikationsprotokoll för att interagera med PLC-system och SCADA-system. Detta är relevant för projektet eftersom OPC UA är ett standardiserat och säkert protokoll med bred interoperabilitet och användning inom industriell automation. Det möjliggör kompatibilitet med olika tillverkares system vilket ökar sannolikheten att verktyget kan fortsätta vara relevant i framtiden.

2. Testverktyget ska fungera oberoende av specifika PLC- eller SCADA-plattformar för att minska beroendet av leverantörsspecifika lösningar, öka flexibilitet och göra verktyget användbart i flera områden.

3. Verktyget ska kunna hantera testning av system med flera styrenheter. Då moderna industriella styrsystem ofta består av flera sammankopplade enheter är det av hög prioritet att verktyget utvecklas med hänsyn till detta. Ambitionen för projektet är att det inte ska finnas

någon gräns (inom rimlighet) för hur många styrenheter ett SUT kan innehålla, men ett krav ställs vid minst två styrenheter.

4. Verktyget ska ge en felrapportering som hjälper användaren att härleda vilken specifik tagg i systemet som givit ett felutslag under ett testfall. Detta underlättar felsökning för användaren som då snabbt kan särskilja vilken av alla taggar i ett testfall som felar.

5. Verktyget ska stödja testning av signaler av datatyperna BOOL och INT eftersom dessa är grundläggande och vanligt förekommande datatyper i styrsystem. De blir därför kritiska för att täcka de mest vanligt förekommande signalerna i ett styrsystem.

3.2 Pythonbibliotek

För datahantering i Excel och kommunikation till server/PLC används ett antal open source bibliotek i utvecklingsprocessen:

Asyncua är ett bibliotek som möjliggör full implementation av OPC UA-protokollet i Python. Asyncua har utvecklats från det tidigare OPC UA-biblioteket vilket gör det möjligt att via Python ansluta till en server och genomsöka datastrukturer samt att läsa och skriva till dessa.

Openpyxl är ett bibliotek som gör det möjligt att läsa och skriva Excel-filer i formaten .xlsx, .xlsm, .xltx och .xltm. Det är särskilt användbart för att automatisera arbetsuppgifter i Excel och för att integrera Excel-funktionalitet i Python-baserade applikationer [28]. Openpyxl låter användaren skapa och redigera Excelfiler genom att skapa blad, skriva data och formatera celler.

Pandas är ett bibliotek som används för att analysera och bearbeta data. Det erbjuder flexibla datastrukturer och verktyg som gör det enkelt att arbeta med strukturerad data på ett effektivt och intuitivt sätt.

Questionary används för att låta användaren interagera med och ge input till program via terminalen. Genom ett antal olika funktioner som biblioteket erbjuder kan användaren besvara frågor genom textinmatning, ja/nej-frågor och selektion i menyer av valbara alternativ. Detta bibliotek används för att skapa ett användargränssnitt med vilket användaren kan interagera med testverktyget. Användargränssnittet beskrivs fortsättningsvis i kapitel 5.1.5.

3.3 Utformning av testprotokoll

För att strukturera upp testning och hålla ordning på de olika signalerna som kan ingå vid testning upprättas ett testprotokoll. Testprotokollet specificerar indata, förväntad utdata samt dokumenterar de faktiska resultaten från testningen. Utformningen av testprotokollet baseras på ett FAT, och ett av dess teststeg kan ses i Figur 3.1 nedan.

Steg	Beskrivning	Förväntat resultat	Händelse	Sign
85	I SCADA Aktivera ”Stäng bom” KomponentID.OT	Kontroll SIMIT A: alla RBL i färdriktning aktiveras blinkandes synkroniserat B: ”bom stänger” B: KomponentID.bomlykta blinkandes C: ”bom stängd” C: KomponentID.bomlykta fast	ok	

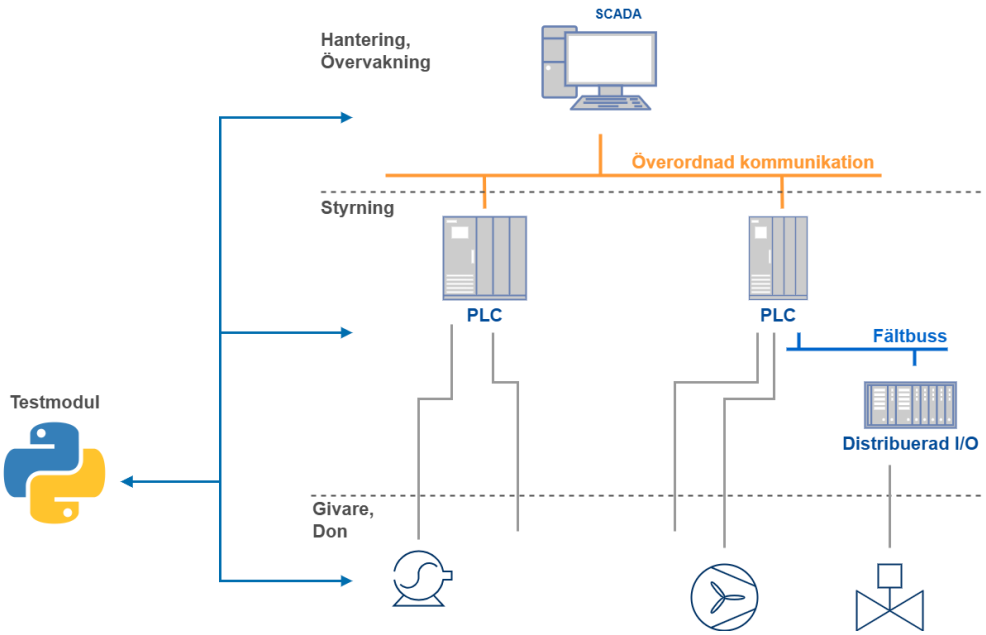
Figur 3.1: Utdrag från ett FAT.

Ur FAT-utdraget kan ses hur teststeg nr 85 har konstruerats genom att en tagg ”KomponentID.OT” som ska stänga en bom aktiveras, och att ett antal förväntade resultat ska ske som en följd av detta. Exempelvis ska ett rinnande blinkande ljus aktiveras, och bommen ska initiera stängning. Dessa kontrolleras av personal och om alla signaler för ett teststeg aktiverats på korrekt sätt så skriver användaren att teststeget är ”ok” under ”Händelse” i protokollet och signerar.

För det automatiska testverktyget upprättas ett liknande dokument i Excel, där ett standardiserat testprotokoll autogenereras. I detta dokument ska användaren ha möjlighet att skriva in teststeg och koppla upp vilka taggar som ska skrivas till och vilka som ska avläsas. När detta väl är gjort körs resterande del av testprocessen automatiskt, och testresultaten skrivs för varje tagg i testprotokollet.

3.4 Övergripande kommunikation med OPC UA

För att kommunikationen ska fungera smidigt krävs ett kommunikationsprotokoll som har brett stöd från flera olika fabrikat. Därför används OPC UA som ger stor flexibilitet i vilka styrenheter testverktyget kan kommunicera med, vilket illustreras i Figur 3.2.



Figur 3.2: Visualisering över hur testverktyget kan interagera med olika enheter samtidigt oavsett dess position i hierarkin.

Figuren illustrerar tanken att testverktyget ska kunna kopplas upp mot samtliga enheter samtidigt, oavsett deras position i topologin. Det ska alltså gå att exempelvis ge en styrsignal för att starta en motor i SCADA, sedan kontrollera så att utgången från PLC-systemet blir aktiv, att den sedan får kontaktorsvar och att SCADA till sist får information om att kontaktorn är dragen och att motorn har gått upp i varv. Anledningen till detta är att det exempelvis ska kunna ge en styrsignal i SCADA och att sedan kunna kontrollera rätt funktion genom att övervaka att till exempel rätt utgång aktiveras på PLC-systemet.

4 SYSTEM FÖR VERIFIERING AV LÖSNING

För att testa av det program som ska utvecklas används en iterativ process där testverktyget körs mot ett styrsystem. Komplexiteten på styrsystemet ökar ju mer utvecklat testverktyget är. De verktyg som används vid test och verifiering är KepServerEX som OPC UA-server och en Siemens S7-1500 PLC med program för automatisk styrning av en vägbom. Detta program har utvecklats av Granitor åt en kund, och används nu inom projektet för att testa mot ett verkligt system.

Till en början används enbart en OPC UA-server som körs i KepServerEX. Denna server innehåller 3 taggar som används för att göra enkla tester på de programfunktioner som utvecklats, exempelvis läsning och skrivning till en tagg i taget eller för test av tagginventeringen.

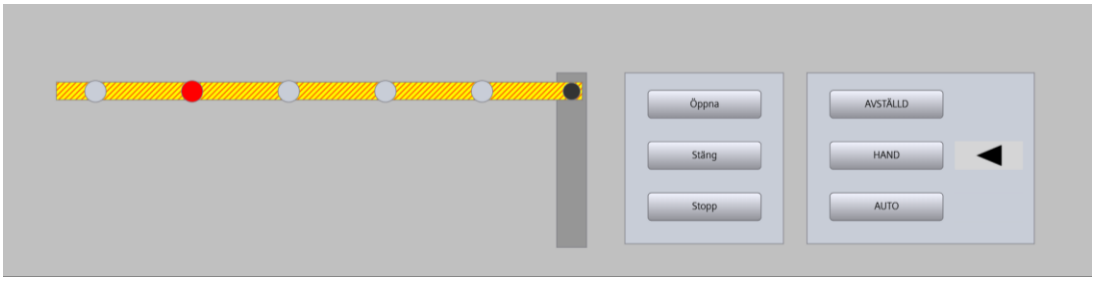
För att genomföra verifiering av testverktyget har ett av testprotokollen som Granitor skapat för bom-styrning använts som grund för att skapa ett testprotokoll som kan genomföras automatiskt. Figur 4.1 visar de första stegen ur testprotokollet för styrning av en vägbom.

Specifikation			Protokoll	
Steg	Beskrivning	Förväntat resultat	Händelse	Sign
Kontroll av konfliktfel efter att bommen har stängts.				
1	I SCADA Aktivera läge HAND i Manöverpanel KomponentID.OH	Kontroll SCADA: Manöverpanel: Indikering Hand Processbild: Indikering Hand	Ok	
2	I bomstyrning: Öppna manöverpanel och aktivera ”manuell” för KomponentID	Kontroll bomstyrning: ”Manuell” aktiveras	Ok	
3	I SCADA Aktivera ”Stäng bom” KomponentID.OT	Kontroll bomstyrning: A: ”bom stänger” aktiveras C: ”bom stängd” aktiveras C: ”Rinandeljus” aktiveras Kontroll SCADA: Manöverpanel: B: ”Bom Stänger” aktiveras B: ”Bom Öppen” deaktiveras C: ”Bom Stängd” aktiveras C: ” Rinandeljus” aktiveras Processbild: B: ”Bom Stänger” aktiveras B: ”Bomlykta Tänd” aktiveras C: ”Bom Stängd” aktiveras C: ” Rinandeljus” deaktiveras	Ok	
4	I SCADA Deaktivera ” Rinandeljus” KomponentID.OF3	Kontroll bomstyrning: Rinandeljus behåller aktiverat läge Kontroll SCADA: Manöverpane: A: ” Rinandeljus” behåller aktiverat läge Processbild: A: ” Rinandeljus” behåller aktiverat läge	Ok	

Figur 4.1: Del ur testprotokoll för styrning av vägbom.

I de befintliga protokollen finns genomförandet beskrivet under kolumn ”beskrivning”. De förväntade resultaten för de manuella testen i Figur 4.1 är grupperade med bokstäver för att indikera i vilken ordning saker förväntas hända. Förväntade resultat med samma bokstav förväntas hända samtidigt. Testprotokollet för manuella tester görs sedan om till ett protokoll som kan genomföras automatiskt.

Vid tester finns inte alltid den fysiska utrustningen tillgänglig. Då används ofta vad som kallas för en digital tvilling. Detta är en virtuell visualisering av anläggningen, som kan styras av det verkliga styrsystemet. Vid tester brukar även SCADA-systemet finnas tillgängligt. På grund av yttre omständigheter fanns varken den digitala tvillingen eller SCADA tillgängligt för projektet. Därför utvecklades en digital tvilling i PLC-systemet med TIA-portal för att visuellt se händelseförloppet när ett automattest genomförs. Den digitala tvillingen syns i Figur 4.2.



Figur 4.2: Digital tvilling av vägbom.

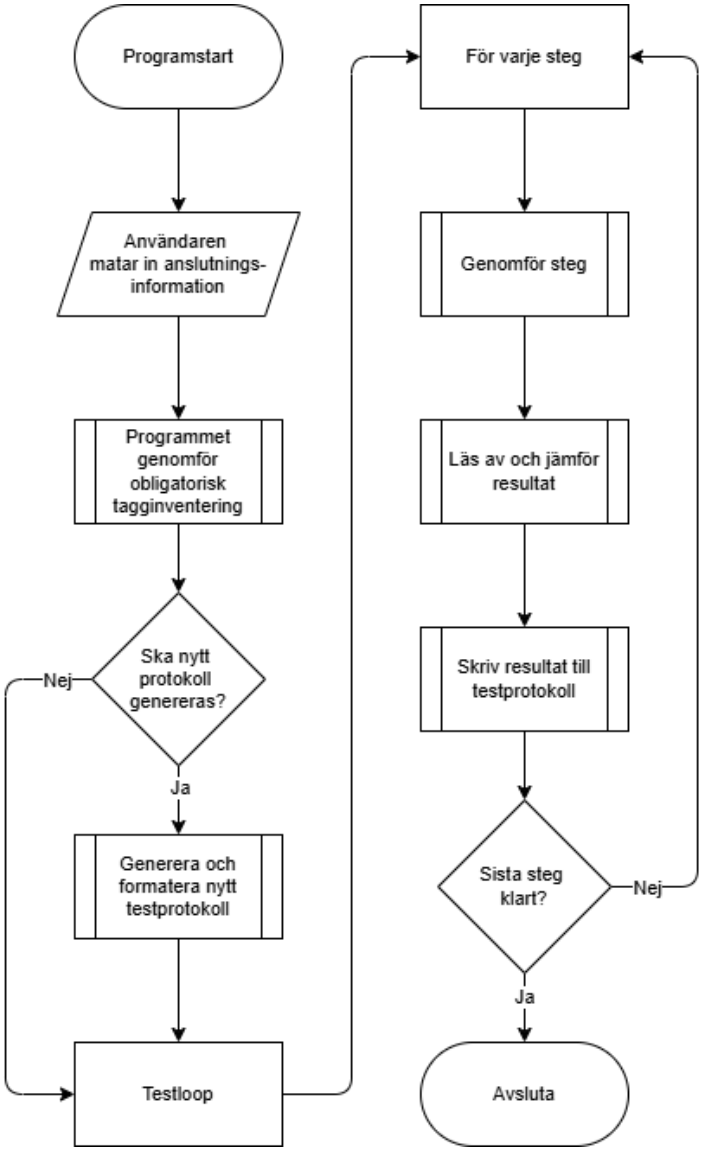
Visualiseringen har animering för öppning och stängning samt har animerade röda lampor. Det finns en meny med knappar för att öppna, stänga och stoppa bommen, och en annan meny för att välja bommens driftläge. Aktuellt driftläge visas med en pil intill knappen för respektive driftlägesval.

5 RESULTAT

Detta kapitel redovisar den lösning som utvecklats under projektet. En översiktlig bild av testverktygets uppbyggnad och dess generella funktionalitet presenteras först. Därefter beskrivs och illustreras testverktygets huvudsakliga funktioner mer ingående.

5.1 Programuppbyggnad

Testverktygets utveckling kan delas upp i olika funktionella delar som alla behövs för ett användbart automationsverktyg. Programmets huvudsakliga komponenter utgörs av en systeminventering, som hämtar samtliga taggar i SUT, generering av testprotokoll, läs- och skrivfunktioner vid körning av test samt ett användargränssnitt som kopplar samman de individuella komponenterna. Figur 5.1 beskriver det övergripande programflödet.



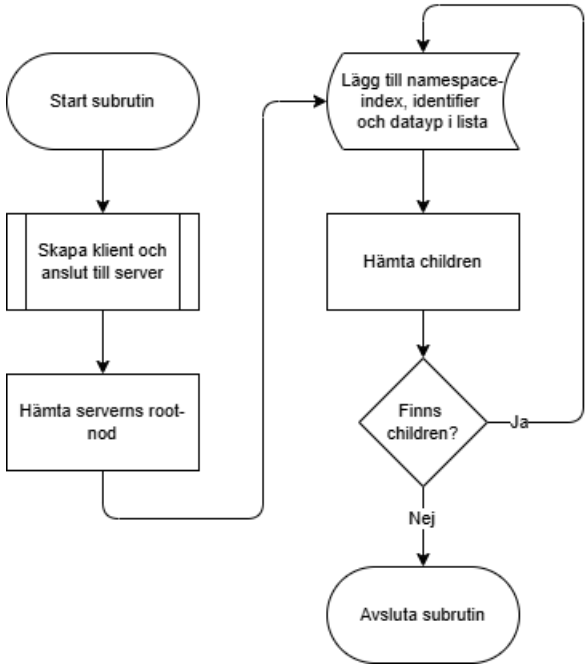
Figur 5.1: Övergripande händelseförlopp för testverktyg.

Programmet initieras med inmatning av IP-adress och port för de enheter som ingår i SUT. Därefter inventeras samtliga taggar från de anslutna enheterna för att användaren ska få

information om samtliga taggar som finns i SUT via testprotokollet som genererats i Microsoft Excel. Det blir därefter användarens ansvar att mata in arkitekturen för de testfall som ska köras och ge information om de taggar som ska skrivas till, samt vilka taggar som ska läsas, och vilka värden som är förväntade. Detta moment i testprocessen är det mest manuellt arbetsintensiva och tidskrävande. När användaren är färdig med detta kan önskade testfall genomföras automatiskt genom skrivning och läsning till inmatade taggar i SUT. Testresultaten skrivs därefter per automatik till testprotokollet i Excel, varpå nästa testfall körs till dess att alla testfall är genomförda.

5.1.1 Inventering av taggar

För att test ska kunna genomföras behöver programmet hämta in information om varje nod som finns på de anslutna servrarna, exempelvis vilken datatyp noden har och vilken IP-adress den finns tillgänglig på. För att hämta denna information genomförs en inventering av alla noder på servrarna. Figur 5.2 beskriver på en förenklad form flödet för inventeringen.



Figur 5.2: Förenklat flödesdiagram för inventering.

Programmet inventerar en server i taget och genomför då en rekursiv sökning av adressutrymmet på servern med start i root- eller objects-noden, detta garanterar att varje nod kommer med i sökningen. För varje nod sparas sedan IP-adressen, identifier, datatyp och namespace-index i ett register. Detta register används sedan vid läsning och skrivning till servern.

5.1.2 Generering av testprotokoll i Excel

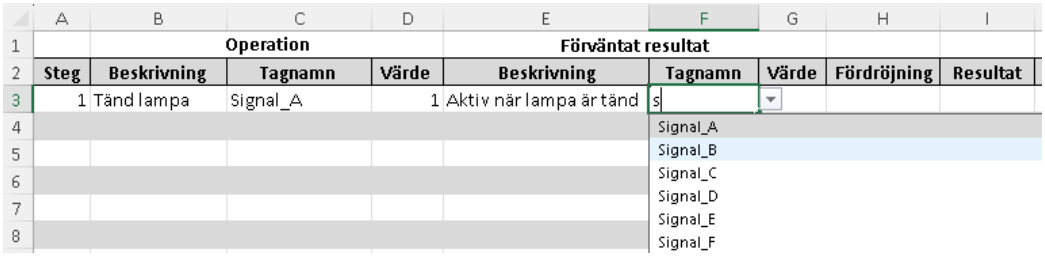
För att en testoperatör ska kunna mata in de korrekta skriv- och läsvärdena för testfallen, alltså det värde som skrivs till för att initiera testet respektive det förväntade resultat som ska avläsas, används Microsoft Excel för att generera ett testprotokoll. Ett standardiserat testprotokoll genereras därför enligt Figur 5.3 nedan.

	A	B	C	D	E	F	G	H	I
1		Operation			Förväntat resultat				
2	Steg	Beskrivning	Tagnamn	Värde	Beskrivning	Tagnamn	Värde	Fördröjning	Resultat
3									
4									
5									
6									
7									
8									
9									

Figur 5.3: Genererat tomt testprotokoll.

De tre kolumnerna under ”Operation” berör den tagg som ett värde skall skrivas till och de tre kolumnerna under ”Förväntat resultat” berör de taggar som ska läsas efter att en operation genomförts. I kolumnen ”Steg” skrivs en siffra med början på 1 och fortsätter uppåt för önskat antal testfall. ”Beskrivning” är en kort beskrivning av den operation som genomförs i testfallet, denna har ingen påverkan på testet och kan lämnas tom om så önskas. ”Tagnamn” är den identifier som importerats med hjälp av OPC UA, följt av det värde man vill ställa taggen till.

Efter att inventering av SUT genomförts så har IP-adress och identifier för varje nod importerats till Excel och en funktion applicerar datavalideringsinställningar för att testoperatören enkelt ska kunna välja input-signal och förväntade output-signaler för respektive testfall. Under kolumnen ”Tagnamn” väljs aktuella taggar enkelt med hjälp av rullmenyer som kan ses i Figur 5.4. Ett steg kan avläsa ett obegränsat antal taggar under ”Förväntat resultat”.



	A	B	C	D	E	F	G	H	I
1		Operation			Förväntat resultat				
2	Steg	Beskrivning	Tagnamn	Värde	Beskrivning	Tagnamn	Värde	Fördröjning	Resultat
3	1	Tänd lampa	Signal_A	1	Aktiv när lampa är tänd	s			
4						Signal_A			
5						Signal_B			
6						Signal_C			
7						Signal_D			
8						Signal_E			
						Signal_F			

Figur 5.4: Användaren fyller i skriv- och läsinstruktioner för olika teststeg.

När användaren skrivit in det antal steg och dess förknippade skriv- och lässignaler sparas filen och den automatiserade testprocessen kan köras. Under kolumnen ”Fördröjning” matar användaren in efter hur lång tid (i sekunder) som taggen ska avläsas. Som standard behövs en liten fördröjning innan avläsning, eftersom skrivning till en tagg kan ta några millisekunder. Dessutom kan avläsningen i vissa fall avse en signal från en komponent vars värde inte uppdateras omedelbart, utan först efter en viss fördröjning. Till exempel kan en bom som får en styrsignal att stängas ta några sekunder innan den faktiskt når stängt läge och givaren indikerar detta.

När användaren väljer att köra testerna i användargränssnittet kommer aktuella insignaler skrivas till SUT och aktuella utsignaler avläsas efter den inmatade tidsfördröjningen. Testresultat skrivs därefter per automatik för varje tagg i testprotokollet. Ett genomfört test visas nedan i Figur 5.5.

	Operation			Förväntat resultat				
Steg	Beskrivning	Tagnamn	Värde	Beskrivning	Tagnamn	Värde	Fördröjning	Resultat
1	Tänd lampa	Signal_A	1	Aktiv när lampa är tänd	Signal_B	1	1	Ok
				Aktiv vid konfliktfel	Signal_C	0	1	Ok
2	Stäng bom	Signal_D	1	Sensor bom stängd	Signal_E	1	10	Ej Ok
				Aktiv när bom kör	Signal_F	1	1	Ok
				Sensor bom öppen	Signal_G	0	10	Ok

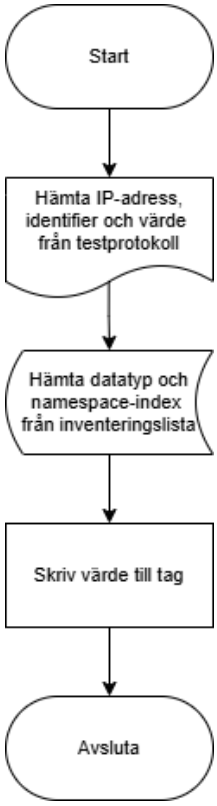
Figur 5.5: Exempel på resultat efter körning av test med två teststeg.

När de önskade stegen skrivits in med tillhörande taggnamn, värden samt fördröjningar och den automatiska testningen genomförts, skrivs testresultaten under kolumnen ”Resultat” längst till höger. I detta specifika fall blev alla taggar godkända förutom Signal_E, vilket skulle indikera att denna del av SUT behöver ses över.

5.1.3 Läs- och skrivfunktioner med OPC UA

För att tester ska kunna genomföras krävs att programmet kan skriva till och läsa från OPC UA-servern. Därför har två funktioner skapats för att göra detta.

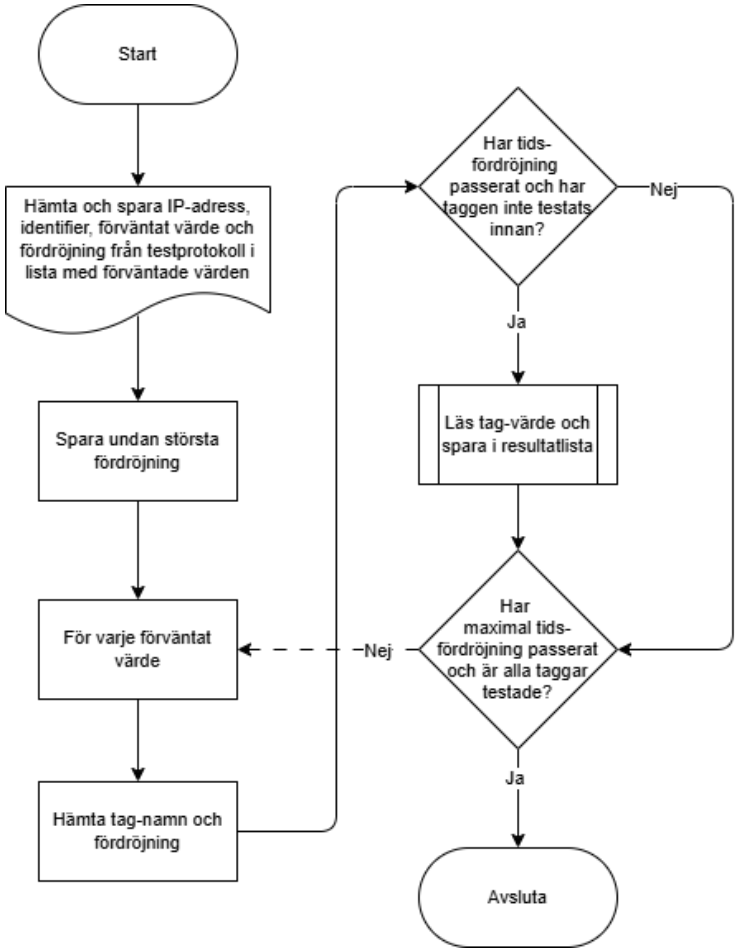
För att skriva ett värde till en nod på en OPC UA-server behöver programmet veta vilken server taggen ligger på, dess identifier och datatyp samt vilket namespace den ligger i. Programflödet för att skriva till en nod beskrivs i Figur 5.6.



Figur 5.6: Flödesschema för skrivning till nod.

Det första som händer är att programmet hämtar IP-adress, identifier och önskat värde för taggen från testprotokollet. Därefter hämtas datatyp och namespace-index genom att söka med IP-adress och identifier i det register som skapats vid inventering. Sist skrivs det önskade värdet till den nod som specificerats i testprotokollet.

De taggar användaren vill läsa av hämtas från testprotokollet och läses sedan av i ordning utifrån de specificerade fördröjningarna. Detta upprepas för varje steg i testet. Figur 5.7 visar flödesdiagram för läsprocessen.

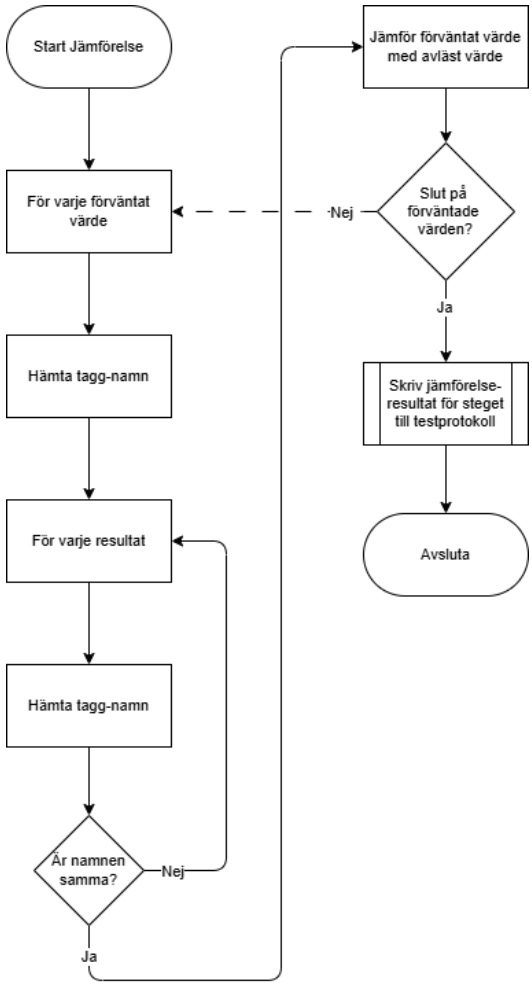


Figur 5.7: Flödesdiagram för läsning av taggar.

Programmet börjar med att hämta in alla taggar för det aktuella steget. Därefter stegas taggarna igenom kontinuerligt, om den för taggen specificerade tidsfördröjningen har gått sedan steget påbörjades läses taggen av och den klassas som testad. Funktionen returnerar ett register som kopplar varje avlästa värde till en IP-adress och ett tagg-namn.

5.1.4 Jämförelse av avlästa och förväntade värden

Efter att avläsning för steget är gjord skall de avlästa värdena jämföras med de önskade värdena. De förväntade värdena finns i ett register. Varje förväntat värde är i registret kopplat till en IP-adress och ett tagg-namn. Flödet för denna visas i Figur 5.8.

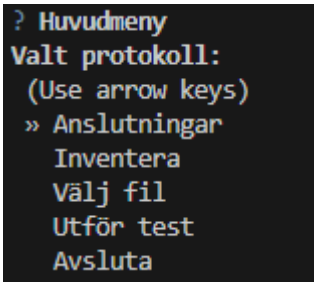


Figur 5.8: Flödesdiagram för jämförelsefunktion.

Denna funktion itererar igenom listan med förväntade värden. Varje förväntat värde matchas med ett avläst värde genom att jämföra namn och IP-adress på förväntat och avläst värde. Därefter jämförs det önskade värdet med det avlästa. Om dessa är lika skrivs ”OK” till testprotokollet, annars skrivs ”Ej OK”.

5.1.5 Användargränssnitt

Vid utveckling av gränssnittet gjordes avvägningar kring användarvänlighet, funktionalitet och utvecklingstid. Bedömningen gjordes att ett menyträd i terminalen ger tillräcklig användarvänlighet, hög flexibilitet och kort utvecklingstid. Gränssnittet har utvecklats med användning av questionary-biblioteket beskrivet i avsnitt 3.2. Figur 5.9 visar en skärmbild på hur huvudmenyn ser ut.



Figur 5.9: Skärmbild från användargränssnittet.

Menyvalen tar antingen användaren till en ny meny, eller så startar en del av programmet. Under exempelvis ”Anslutningar” finns möjligheten för användaren att lägga till, ta bort och lista upp anslutna enheter och vid tryck på ”Inventera” genomför programmet tagginventering.

5.2 Resultat av automatiska tester på vägbom

Vid applicering av det automatiska testverktyget på det testsystem som beskrivs i avsnitt 4 visar testverktyget stor potential. Testverktyget utför tester exakt som användaren skrivit in dem och ger därefter tillförlitliga testresultat. Vid tester med KepServer uppvisar testverktyget förmåga att utföra tester över olika system vilket därmed antyder att verktyget är plattformsoberoende.

Vid utvärdering av tester tillsammans med handledare konstateras att det automatiska testverktyg som utvecklats har stor potential till att spara tid.

6 DISKUSSION

Projektet har visat att det är möjligt att utveckla ett plattformsoberoende verktyg för automatiserade styrsystemtester med hjälp av OPC UA. Verktyget har utvecklats i Python, med hjälp av ett antal externa bibliotek. Styrsystemstester har genomförts mot en fysisk och virtuell PLC. För att utvärdera verktygets kapacitet att spara tid genomfördes automatiserade tester i ett styrsystemsscenario. Signaler av datatypen BOOL och INT har testats och fungerar. Resultaten från testningen visar att ett automatiserat testverktyg sannolikt kan vara ett användbart hjälpmedel vid arbete med styrsystemstestning. De testfall som utvärderats i ett styrsystemsscenario visar att testverktyget har potential att spara tid. Med det sagt behöver användaren också förstå att testverktyget har en del begränsningar vilka diskuteras mer i detta kapitel.

6.1 Systembegränsningar

Vid implementation av testverktyget och användning i verkliga systemmiljöer är det viktigt att förstå vissa begränsningar vid testningen. Ett SUT är i praktiken ofta beroende av ett antal externa komponenter. Det kan vara givare, aktuatorer eller kommunikationslänkar som av olika anledningar kan vara felkällor. Externa faktorer, tekniska fel och allmänt slitage kan göra att komponenter helt enkelt inte beter sig som förväntat. Eftersom testverktyget oftast tolkar systemets tillstånd utifrån insamlade signaler och jämför dessa mot förväntade värden, finns risken för *falskt positiva* eller *falskt negativa* testresultat. Ett exempel är när en givare levererar ett korrekt värde ur signalperspektiv, men värdet inte motsvarar verkligheten, till exempel om en trycksensor är felaktigt inställd. Testverktyget kommer då att godkänna testet trots att systemets verkliga funktionalitet är felande. För att hantera denna typ av begränsning, krävs i många fall en kompletterande visuell eller manuell kontroll för att säkerställa att systemet fungerar korrekt i sin helhet. Det blir därmed också viktigt för en testoperatör att tydligt dokumentera vilka delar av ett SUT som omfattas av den automatiserade testningen, samt vilka kompletterande kontroller som är nödvändiga för att validera systemets totala funktion. En annan aspekt av testverktygets begränsningar är att lösningen i sin nuvarande konstruktion tar en ögonblicksbild av en taggs värde snarare än att göra kontinuerlig övervakning. Detta är i de allra flesta fall inte något problem, men kan göra det svårt att testa signaler som ska bibehålla sitt värde över en viss tid. Exempelvis signal för att en bom öppnar, denna ska bli sann så fort bommen börjar öppna och ska bli falsk när bommen når öppet läge.

6.2 Hållbarhetsaspekter

Automatiserad testning av styrsystem kan bidra till en mer hållbar verksamhet genom att resursanvändning effektiviseras, risker minskas i farliga arbetsmiljöer, samt att personalens välbefinnande och arbetsglädje ökar. Nedan belyses några viktiga hållbarhetsaspekter länkade till automatiserad testning.

Effektivisering, kvalitet och kostnadsbesparingar

Automatiserade testverktyg kan spela en central roll i att effektivisera testprocesser inom styrsystem, särskilt vid upprepade tester och i komplexa systemmiljöer. En viktig fördel med

automatiserad testning är möjligheten att minska repetitiva arbetsuppgifter. Manuell testning är ofta både tidskrävande och monoton, vilket kan påverka arbetsglädjen negativt och därmed även produktiviteten. Genom att ersätta dessa återkommande moment med automatiserade processer frigörs personalens tid till mer kvalificerade och kreativa arbetsuppgifter. Detta kan leda till ökad arbetstillfredsställelse och bättre resursutnyttjande. Dessutom kan automation, om den implementeras korrekt, minska risken för fel som uppstår till följd av den mänskliga faktorn.

Vid regressionstestning av styrsystem kan automatisering vara särskilt värdefull. Som tidigare nämnt syftar regressionstester till att säkerställa att uppdateringar i styrlogiken inte skapar oönskade bieffekter i tidigare funktionalitet. Detta kräver ofta noggrant definierade och upprepade testfall, vilket manuellt är både resurskrävande och känsligt för misstag. Ett automatiskt testverktyg möjliggör att dessa testcykler kan köras oftare, snabbare och med större täckning, vilket förbättrar systemets tillförlitlighet och minskar arbetsbelastningen på testpersonalen.

Utöver att effektivisera testningsarbetet har automatiserade testverktyg även potential att sänka personalkostnaderna. I takt med att styrsystem blir mer avancerade och hanterar fler signaler och tillstånd, ökar även tidsåtgången för testning. Automatiserade testverktyg är skalbara och väl lämpade för att hantera de stora volymer enhetstester som ofta krävs i komplexa system. Genom att minska behovet av manuell arbetsinsats kan sådana verktyg bidra till lägre driftkostnader. Samtidigt främjar de en högre grad av standardisering i testförfarandet, vilket stärker spårbarhet, kontinuitet och långsiktig kvalitet i testprocessen.

Säkrare testning av system i högrisk-miljö

Automatisk testning kan spela en särskilt viktig roll vid implementering och verifiering av styrsystem i miljöer där mänsklig åtkomst är begränsad eller där säkerhetsriskerna är höga, såsom i gruvdrift, kärnkraftsanläggningar eller undervattenssystem. Styrsystem i denna typ av miljö kan kräva särskilt hög noggrannhet vid testning för att förhindra olycka. Genom att använda automatiska testverktyg kan tester köras upprepade gånger utan fysisk närvaro, vilket minskar risken för fel som kan uppstå på grund av mänsklig trötthet eller bristande överblick. Med det sagt kan automatiserad testning sannolikt inte ersätta manuell testning helt i säkerhetskritiska miljöer, men den kan användas som ett effektivt komplement.

6.3 Avslutande reflektioner

Trots många fördelar med automatiserad testning finns det utmaningar med att införa den på ett heltäckande sätt. Automatiska testverktyg är särskilt användbara vid kvantitativa och repetitiva tester, där de kan bidra till både ökad noggrannhet och tidsbesparing. För att säkerställa systemets funktion inför driftsättning bör automatiska tester dock alltid kompletteras med manuella tester. För att undvika felaktig användning är det avgörande att användaren förstår verktygets funktion och vilka delar av systemet som faktiskt testas automatiskt. Annars finns en risk för övertro på verktygets täckning och att andra viktiga kvalitetskontroller därmed nedprioriteras.

6.4 Rekommendationer för fortsatt arbete

Ett tydligt behov för att vidare effektivisera ett testverktyg är att skapa ett grafiskt användargränssnitt som eliminerar behovet av att öppna och stänga Excel manuellt, vilket skulle förenkla arbetsflödet. Att skapa en mekanism för felhantering i Excel, som ger en varning om användaren skulle mata in ogiltiga taggnamn är också aktuellt. Vidare föreslås ett alternativ till fördröjningsbaserade tagg-avläsningar i form av en prenumerationsmodell för taggar som kräver kontinuerlig övervakning. Det bör även utvecklas möjligheter att testa olika intervall för motorer och liknande komponenter där varvtal ska avläsas. Slutligen är det viktigt att verktyget genomgår mer omfattande testning på större och mer komplexa testobjekt för att undersöka robusthet och tillförlitlighet mer utförligt i praktiska sammanhang.

7 Referenser

[1] K. Wiklund, S. Eldh, D. Sundmark och K. Lundqvist, ”Impediments for software test automation: A systematic literature review,” *Software Testing Verification and Reliability,* vol. 27, nr 8, Dec 2017.

[2] BBC, ”Client-servers and peer-to-peer networks,” 2025. [Online]. Available: https://www.bbc.co.uk/bitesize/guides/zvspfcw/revision/4. [Använd 16 april 2025].

[3] S. Vitturi, C. Zunino och T. Sauter, ”Industrial Communication Systems and Their Future Challenges: Next-Generation Ethernet, IIoT, and 5G,” *Proceedings of the IEEE,* vol. 107, nr 6, pp. 944-961, 2019.

[4] OPC Foundation, ”What is OPC?,” 2025. [Online]. Available: https://opcfoundation.org/about/what-is-opc/. [Använd 14 april 2025].

[5] D. Eidenskog och B. Lindahl, ”NCS3 – Industriella protokoll i Sverige,” Totalförsvarets Forskningsinstitut, 06 juli 2017. [Online]. Available: https://www.foi.se/rapportsammanfattning?reportNo=FOI-R--4438--SE. [Använd 14 april 2025].

[6] OPC Foundation, ”OPC 10000-100 Devices,” 03 11 2022. [Online]. Available: https://opcfoundation.org/developer-tools/documents/view/197. [Använd 28 april 2025].

[7] OPC Foundation, ”OPC 10000-8 UA Part 8: DataAccess,” 29 november 2024. [Online]. Available: https://opcfoundation.org/developer-tools/documents/view/165. [Använd 28 april 2025].

[8] Siemens, ”Totally Integrated Automation Portal,” 2025. [Online]. Available: https://www.siemens.com/global/en/products/automation/industry-software/automation-software/tia-portal.html. [Använd 28 april 2025].

[9] PTC, ”Kepware Industrial Connectivity Solutions,” 2025. [Online]. Available: https://www.ptc.com/en/products/kepware. [Använd 15 april 2025].

[10] Novotek Sverige, ”KepServerEX,” 2025. [Online]. Available: https://www.novotek.com/se/losningar-och-produkter/opc-industriell-kommunikation/kepserverex/. [Använd 15 april 2025].

[11] E. O. Travis, ”Python for Scientific Computing,” *Computing in Science and Engineering,* vol. 9, nr 3, pp. 10-20, 2007.

[12] T. Pham, A. P. Boone och M. K. Ngo, ”Heuristic Evaluation of Supervisory Control and Data Acquisition (SCADA) Displays,” i *Proceedings of the Human Factors and Ergonomics Society*, San Diego, CA, United States, 2023.

[13] A. Nechibvute och H. D. Mafukidze, ”Integration of SCADA and Industrial IoT: Opportunities and Challenges,” *Institution of Electronics and Telecommunication Engineers,* vol. 41, nr 3, pp. 312-325, 2024.

[14] G. J. Myers, T. Badgett och C. Sandler, The Art of Software Testing, Hoboken, New Jersey: John Wiley & Sons, Inc., 2012.

[15] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz och S. Yoo, ”The oracle problem in software testing: A survey,” vol. 41, nr 5, pp. 507-525, 2015.

[16] D. Coppit och J. M. Haddox-Schatz, ”On the use of specification-bassed asserions as test oracles,” i *2005 29th Annual IEEE/NASA Software Engineering Workshop*, Greenbelt, MD, USA, 2005.

[17] N. Wild och H. Lichter, ”Unit Test Based Component Integration Testing,” i *Proceedings - Asia-Pacific Software Engineering Conference*, Seoul, 2023.

[18] The Institute of Electrical and Electronics Engineers, IEEE Standard Glossary Of Software Engineering Terminology, 1990.

[19] M. Zielinski och R. Groenboom, ”Using advanced code analysis for boosting unit test creation,” i *14th International Conference on Software Testing, Verification and Validation Workshops*, Virtual, Porto de Galinhas, 2021.

[20] GeeksforGeeks, ”Unit Testing - Software Testing,” GeeksforGeeks, 2025.

[21] S. Banitaan, K. E. Nygard, M. Alenezi och K. Magel, ”Towards test focus selection for integration testing using method level software metrics,” i *10th International Conference on Information Technology: New Generations*, Fargo, North Dakota, United States, 2013.

[22] A. Fuchs och V. Von Hof, ”Improving integration testing of web service by propagating symbolic constraint test artifacts spanning multiple software projects,” i *International Conference on Software Engineering and Knowledge Engineering*, Münster, Germany, 2018.

[23] D. Qingfeng och Z. Huijuan, ”An Effective Design Method of System Function Test Cases,” i *2011 International Conference on Computer and Management*, Wuhan, China, 2011.

[24] R. Gupta och V. Jaglan, ”Coverage Based Test Suite Minimization using UML Behavioural Diagrams,” i *International Conference on Future of Engineering Systems and Technologies*, Greater Noida, India, 2020.

[25] C. Wang, F. Pastore, A. Goknil och L. C. Briand, ”Automatic Generation of Acceptance Test Cases from Use Case Specifications: An NLP-Based Approach,” *Transactions on Software Engineering,* vol. 48, nr 2, pp. 585-616, 2022.

[26] J. Hartmann och D. J. Robson, ”Approaches to regression testing,” i *Conference on Software Maintenance*, Durham, England, 1988.

[27] D. Huizinga och A. Kolawa, Automated Defect Prevention: Best Practices in Software Management, Hoboken, New Jersey: John Wiley & Sons, 2007.

[28] S. Hou, M. Hou och Z. Xiong, ”Python-based Remote Server Data Collection Program for VASP,” i *4th IEEE International Conference on Electronic Technology, Communication and Information*, Shijiazhuang, China, 2024.

[29] Y. Bhatt och P. Pahade, ”Application of Python Programming,” i *Information and Communication Technology for Competitive Strategies*, Singapore, Springer, 2021, p. 849.

[30] A. Staaby, K. Steenbjerg Hansen och T.-M. Grønli, ”Automation of Routine Work: A Case Study of Employees' Experiences of Work Meaningfulness,” i *Proceedings of the 54th Hawaii International Conference on System Sciences*, Hawaii, USA, 2021.

INSTITUTIONEN FÖR ELEKTROTEKNIK

CHALMERS TEKNISKA HÖGSKOLA

Göteborg, Sverige 2025

www.chalmers.se

CHALMERS