

Auto-Testing of Code



Flamur Breznica

Division of Industrial Electrical Engineering and Automation
Faculty of Engineering, Lund University

LUND UNIVERSITY
FACULTY OF ENGINEERING

DEGREE PROJECT IN INDUSTRIAL ELECTRICAL ENGINEERING AND
AUTOMATION

EIEM01

Auto-testing of code

Author:

Flamur Breznica

Elt12fbr@student.lu.se

Supervisors:

Mats Lilja

Mats.Lilja@hbg.lth.se

Christian Isaksson

Christian.Isaksson@ecolean.se

Examiner:

Gunnar Lindstedt

Gunnar.Lindstedt@iea.lth.se

March 11, 2020

Summary

The topic of this thesis is software in systems of automated machines that produce products in a company. Testing of these systems is crucial for the machines to work as intended. Usually testing is done manually, both while the system is developed and while the production is running. This is called traditional testing. Testing is done to improve the machines program code. The machine needs to do something wrong to fail the test. Certain faults lead to the change of certain parts of the code. Test-cases have to be developed to address different parts of the code. Nowadays testing is usually done completely manually. This thesis will explore the possibilities and the limitations with partly automatic testing, as well as develop a theoretical machine to test.

The thesis was done at a machine company in Helsingborg. The approach is to use an automatic regression testing of the code. Firstly an analysis was done which went through the existing options and refine the approach. One approach was selected and used for the thesis. The main task of the thesis is to facilitate the way of testing the machines for companies, or at least help companies to improve machines, with earlier commissioning and enable distance maintenance. The analysis went through different software aspects and old reports written about automatic testing. Then implementing a testing strategy to a control system. The control system needed to be developed and a theoretical machine were needed to be tested. The next part was to simulate the machine and lastly illustrate the scenario with a pictorial view.

By developing the machine code it was possible to create automatic tests on it, called unit testing. This was combined with a simulation of the machine. This was successfully shown on a human machine interface. By doing this it is possible to save time and money but it will depend on the programmer creating the tests. The simulation was done through software that coded the controller, called TIA portal. In the future a better and more suitable environment could be used for simulation purposes. Unit testing and simulation proved to work well for automatic testing. Remote maintenance and testing requires simulating the entire machine with all of its complex parts. All physical hardware can be simulated.

Simulating and testing on a machine was implemented. The combination is considered to be time and money saving and that there are options for making the tests partly automatic and machines with higher quality.

Keywords: IEC-61131-3, regression testing, automation, automatic, machines, S7 Unit test, unit testing, NUnit, Siemens, TIA Portal, Simulation, SIMIT, automation production systems, simulation, digital twin.

Sammanfattning

I denna rapport diskuteras mjukvara som används för system bestående av automatiska maskiner som producerar produkter i ett företag. Testning är nödvändigt för att maskinerna ska fungera som planerat. Oftast görs testning manuellt, både under utveckling och medan produktionen är igång. Detta kallas för traditionell testning. Testning görs för att förbättra maskinens programkod. För att testen ska bidra med något måste maskinen misslyckas med testen. Vissa fel leder till ändringar i vissa delar av koden. Test-fallen måste därför adressera olika delar av koden. Idag görs oftast testning helt manuellt. Detta examensarbete ska utforska möjligheter och begränsningar för att delvis införa automatiska tester, samt utveckla en teoretisk maskin att testa.

Examensarbetet gjordes på ett maskinföretag i Helsingborg. Metoden var att använda en regressions-testning av koden. Först gjordes en analys som gick igenom nuvarande möjligheter och förfina tillvägagångssättet. En metod valdes och användes för examensarbetet. Huvudsyftet med examensarbetet är att underlätta test av maskinerna för företag, eller åtminstone hjälpa företag att förbättra maskiner, vilket medger tidigare driftsättning och möjlighet till distansunderhåll. Analysen gick igenom olika mjukvaruaspekter och tidigare rapporten skrivna om automatisk testning. Sedan valdes en strategi som skulle implementeras på ett kontrollsystem. Kontrollsystemet behövde utvecklas och en teoretisk maskin behövdes för testning. Nästa steg var att simulera maskinen och illustrera scenariot med en grafisk operatörsbild.

Genom att utveckla maskinkoden var det möjligt att skapa automatiska tester på den, så kallade enhetstest. Detta kombinerades med en simulering av maskinen. Detta visade sig framgångsrikt på ett människa-maskin-gränssnitt. Genom att göra detta är det möjligt att spara tid och pengar men det kommer vara beroende på programmeraren som skapar testerna. Simuleringen gjordes genom mjukvaran som kodar kontrollenheten. Mjukvaran heter TIA portal. I framtiden kan en bättre och mer lämplig miljö användas för simuleringsändamålet. Enhetstestning och simulering visade sig fungera bra för automatisk testning. Fjärrunderhåll och fjärrtestning kräver simulering av hela maskinen med alla dess komplexa delar. All fysisk hårdvara kan simuleras.

Simulering och testningen av maskinen implementerades. Kombinationen anses vara tid- och kostnadsbesparande och det finns möjligheter för att delvis göra tester automatiska och även att få maskiner med högre kvalitet.

Acknowledgements

This thesis was done at Ecolan in Helsingborg. I have received input from all the coworkers at the automation department. They were both great with input but also great to bounce ideas too since the thesis formed. I am also very grateful for the help and this opportunity given to me by Christian Isaksson who made all of this possible, without his contribution none of this would be possible.

I would like to say thanks to have Mats Lilja who accepted to be supervisor in this thesis even though it was not his job. This helped since the thesis was done in Helsingborg and it helped to have a supervisor in the same city. Mats Lilja has been a great supervisor and his help should not be overlooked, really appreciated to have him and discuss ideas when there was a need for it.

Finally, I would like to thank the examiner Gunnar Lindstedt for helping me set up the thesis goals and starting the thesis quickly.

Contents

1	Introduction	1
1.1	Background and purpose	1
1.2	Goals	1
1.3	Presentation of problems	1
1.4	Methodology	1
1.5	Limitations	2
2	Theory	3
2.1	Earlier works	3
2.1.1	How to prioritize tests	3
2.1.2	Evolution of automation production systems	4
2.2	Tools	5
2.2.1	S7 Unit test	6
2.2.2	SIMIT	7
2.3	Model-View-Control	8
2.4	Test automation pyramid	8
3	Machine	10
3.1	Hardware	10
3.2	Programmable logic controller programming	11
3.3	Human machine interface	13
3.4	Simulation	15
3.4.1	TIA Portal	15
3.5	Testing	18
3.5.1	Loading TIA project	22
4	Result	24
4.1	TIA Portal	24
4.2	Unit testing	25
5	Discussion	28
6	Conclusion	32
7	Future work	33
	Bibliography	34
	Appendices	36
A	S7 unit test result	36

1 Introduction

This chapter gives background and presents goals, methodology and limitations for this thesis work.

1.1 Background and purpose

Companies that sell manufacturing machines need to investigate automatic testing. This could improve machines to have a bigger capacity which is mandatory today, this also enables the option to have regular maintenance and this is something every company strives for. While increasing the capacity it is important to maintain the quality and if possible, even raise the bar. To accomplish higher quality and increasing the capacity automated testing is a good way to go since it has the potential to lower the cost and live up to the expectations. It will lower the cost of continuous maintenance since it will not need to shut down machines for testing.

To verify functionality in automation software tests are done on physical machines to run through different parts of the code. For machines that are not available in the workshop, tests need to be performed at the customer sites. This means that all machines that need updates or maintenance and are shipped to the buyer will involve travels. Since this creates costs due to travels and disrupts customer production these updates are kept at a minimum. To minimize tests on site a system to simulate functionality is available in the automation software. But the tests depend on manual work and each test scenario must be run manually, which means that a person needs to stay and observe the scenario for it to pass. The cost is driven up, and even out on site the machines need to be stopped for maintenance and will cost more. If it is possible to write a code that makes distance updates possible and tests them on distance this will lower the costs. Shorter maintenance times equals happier customers.

1.2 Goals

The goal of this thesis is to investigate and use a regression test system that can automatically test code by simulating different scenarios in the machine.

1.3 Presentation of problems

- What methods are used for regression testing in automation software?
- What limitations and possibilities are there with these kinds of tests?
- Limit decision criteria based on one part of the machine and decide on one approach for the implementation of a regression test system.
- Is it time-wise profitable versus physical machine testing?

1.4 Methodology

The thesis needs to start with a pre-study to find out an approach for regression testing automation software and the limitation and possibilities with these kinds of tests. With the help of the pre-study, a suitable limit decision criterion is chosen based on one part of the machine and decision on one approach for the implementation of a regression test system.

On this part of the thesis earlier knowledge of programming is used that has been taught through education. It will be built on linking java-programming courses for industrial applications that are written in IEC-61131-3. It will be needed to create a program to the controller for a filling machine line that has one product on the line at a time, the virtual machine will then be tested. Then a time analysis will be applied to the entire cycle for setting up the tests and then it will be compared to the old traditional way when it needs to be tested on the machines. The tools that will be used through this thesis are Siemens TIA Portal [1] software and the main source for information will be the help functions in the software as well as old related researches.

Automatic testing is a very hot topic and many big companies are developing different tools and are investigating the possibilities for automatic testing since there is a good possibility that this will be very cost-saving and even very environmental preservative since the trips potentially will be fewer. Many different companies are trying to implement this in a user-friendly way. The possibility of automatic testing is that it might make the machines better as well, bugs might be fewer on the commissioning day since the tests are done through the production. The market is craving for stuff like this, so big manufacturers are spending time to develop programs that are meant to be used. To make automatic testing possible it is needed for continuous integration which is a hot topic in software engineering. [7]

1.5 Limitations

To program a machine is very complex and time-consuming. The machine will be a fill machine line with different stations and different tasks. The limitation for the controller software will be that there is only one product on the conveyor of the fill machine line. The simulation will be implemented traditionally and the new simulation software will be put as future work. As for testing an automatic unit testing program will be used. Everything in this thesis is based on simulations, there will be no implementation in real hardware. All tests will be simulated on a simulated controller on the computer.

2 Theory

This chapter will review earlier works done relating this topic where an investigation is done. Then it will go through what tools will be used. After the tools, there will be a review of a couple of a strategy about automatic testing.

2.1 Earlier works

Testing in automation production systems has traditionally been done on the machines which take up time and are expensive. Every update required that the machines needed to be turned off and the update had to be done. This meant that the engineer doing the update got paid but also the operators and different individuals in the factory. To lower these costs, it is possible to do software tests and lower the downtime of machines. While doing this there are many advantages like there is no pressure on the engineer coding them like it is on site, which means the risk for faults will be lower than doing it on site. One example of automatic software testing is given in China where work has been written about automatic software testing where it was done with "Software testing automation framework". [2] This was done by using an open-source framework. To make the automatic testing work it demanded them to create a working program and apply this to a true work environment. In the report they show that it is cost-saving to make the tests automatic. According to the report, several cases had to be created and automatically run several times. This would require a large amount of time for a test engineer to perform manually. This is done on software but is an example of how favorably it can be with automatic tests.

After updates, there is always a certain risk for fault, those are called regression faults. For automatic testing to work in the industry it must be prioritized correctly. To prioritize correctly is not an easy task and when faults show up on site in the industry it is often in pressured positions that are not always easy to prioritize the correct tests that need to be done. If the tests are correctly picked the results will be promising and save both time and reduce the costs. [3] To choose the correct test cases are at least as important as actually executing the tests automatically since the actual tests will have better results. The study was advanced and implemented in the industry testing [4]. Another option to make the tests automatic is to rewrite the tests on a GUI(Graphical User interface) application and move the tests over to the software side. On the software side, there are many testing utilities and possible to get it tested. [5] It is possible to do a model-based testing and test cases, also known as unit-testing. [6] This is done on the Programmable logic controller (PLC) but shows that it is possible to do different test cases that can be used as regression tests.

2.1.1 How to prioritize tests

There are many ways of prioritizing tests, the most common is by going on intuition and experience. This will depend on the person doing the test and if it is done on site, it will be more difficult choosing wisely and most tests done are unnecessary or could have been avoided. Now there is no help for finding and prioritizing the test in automation production systems. [4]

A regression fault is a fault that has shown up when a new function was introduced. In the report written by Ulewicz and Vogel-Heuser, an iterative process is used based on the IEC-61131-3 standard. This is used in most of the automation production systems. By updating and changing the code there is often introduced new errors, bugs or new customer demands in the automation production system. This means that there are faults in the code that keeps reappearing and the testing is done manually on site, often under pressure. The approach made is an iterative process, but in this thesis, there will be some parts selected from the tests that will be used and some will not. In this report, the approach was developed in detailed simulation or formal specifications. The process is made that between every test case made there is a relation between it and the developed control functions. Having relationships with the previously tested versions will give a connection to each other that will help with the testing. By doing this it will be easier to see where the faults origin is. The next step is looking up the differences between the previous version of the program and the possible effects of the implemented changes, need to be identified. Then earlier system tests for the actual version are prioritized. This is based on possible effects of the changes made on the previous step.[4] In essence, it works like the following:

- If the controller has a version, the next version needs to be analyzed on the changes done versus the previous version.
- These changes are then implemented in the tests and earlier tests that relate are prioritized.
- Create a relation between tests and versions.
- The unit tests are set and checked.

This is done by an algorithm made in the report written by Ulewicz and Vogel-Heuser. [4] But in this thesis it will just observe that prioritizing tests are as important as actually making them automatic, and in some cases even more profitable. By prioritizing tests like the article refers, it is easier and an analyzing aspect is implemented. Analyze takes time, but in the long run it is profitable. This is something that is often forgotten when under time pressure and stress, but just by thinking of it will speed up the testing process.

2.1.2 Evolution of automation production systems

There was an article written about the evolution of automation production systems, the challenges it faces and research directions. The article is written by Birgit Vogel-Heusera, Alexander Fay, Ina Schaefer and Matthias Tichy. [7] In this part of the theory a summary of the article will be given, the summary will be written to relate the thesis.

Automation production systems have it all. It has software, electrical parts, hardware and software. This all combined means mechatronics needs to be accounted for. The software opens the possibility for remote updates, where hardware needs to be done on site. These systems are very complex they have a lifetime of 10-20 years, which means that they need to be made for that. This means that they must change under their lifetime and keep up with the innovations. Continuous integration is a hot topic, the goal is to make development faster and also making maintenance regularly. To integrate the hardware and the software is more complicated than in a software setting for phones or computers. The changes in hardware are much slower as well as simulation models of hardware are required for automated integration tests. [7]

These kinds of systems have many challenges. To reach their requirements they need to be adapted or expanded. This is often overlooked when the project's time is running out. The complexity of software in automation is growing as well as the challenges remain to keep up with the evolution. At the moment most manufacturing systems use the programming language IEC 61131-3, the same is used in this thesis. There have been efforts to upgrade the language to object-oriented within this standard, but its version is not fully established in the industry. There is one common technique used in the industry which is clone and own, for example if there is a code that does the same thing. Usually, this is copied and pasted on the code wished for. This is a big challenge in the software, to implement modular software that has a relation is yet to be done. With clone and own, if there are a bug and that part of the code is written in different places. In that case, it is needed to locate them and fix them everywhere. In automation production systems the code is usually old and needs restructuring as well. This also implies to the PLC suppliers. A case study was done on a leading manufacturing company where the code was analyzed, and due to an approach for universal components which includes all variants meant that 80% of the code was useless for many machines. [7]

Design patterns are usual in software engineering to support code modularity and evolution. For example, if one part is coded, another should not be affected. One pattern that emerged from model-based design to automation software is UML. This has different types of controls and guidelines for implementation. There are also common software design patterns that are being used like distributed application, proxy and model-view-controller that were defined and evaluated. But this was done on IEC 61499 programs. The function block diagram is specified in the IEC 61131-3 and applied, which means that it is possible to have hierarchy and implementing design patterns to have modularity. [7]

Changes are mandatory in automation production systems, this can be due to a new law or just that the client has changed their mind. The ultimate solution would be to have software that verified everything while writing it and is a good goal to have in mind. Changing code in one place might affect other parts of the code. If this creates new faults, they are called regression faults. When applying model-based software, the evolution of these systems needs to be supported by correct models. This is a challenge the automation

production system has and can be solved with modular systems. To make modular systems is a big challenge since it has both hardware and software to take account of. But these kinds of modules can then be reused and adopted in the future. These kinds of approaches need to be adapted for automation production systems. [7]

The approaches to address this challenge are digital factories and virtual commissioning techniques, these are based on simulation and virtual controllers. This opens up the possibility to test the systems, but faults that are difficult to detect will not be detected. These faults could be complemented by analysis techniques. Static code analysis is successfully applied to different programming languages and environments, which could be implemented in automation production systems as well and tools for this are not available yet. The benefits are that faults would be highlighted in the software. If automatic verification is compared to simulating the models, automatic verification has a big advantage that it can be applied earlier in the design phase. Manual testing is still dominant since there are few tools to do them automatically. [7]

There is a big variety of approaches for testing. But these do not take the characteristics of PLC software into account. By simulating checking the behavior of the PLC exists. Therefore, to implement this to industrial PLC software is still an open issue. Nevertheless, automatic validation is possible and more powerful if combining different approaches. Siemens has an implementation that supports semi-automatic merging of model fragments to complete product lines. [7]

Using model-driven engineering decreases the time needed for a machine and raises the quality. It is used in the embedded software industry by using the knowledge, it also introduces challenges specific to the production systems. According to the article, three surveys of practice in model-driven engineering shows that tool integration and usability of the tools as well as high effort is needed to get small benefits in the industry. The model approach enables an integration method for software models and the physical model to be used. Automation production systems have to do with physics and that differs from computer science. There have been hybrid models developed to use MATLAB/Simulink to a programming language called CFC for different PLC programming environments. [7]

The report concluded that automation production systems usually have a customer-specific machine or plant. It is important to maintain strong relationships and dependencies between software, automation hardware and mechanics. Evolution in these systems is necessary and may also include unanticipated changes. Solutions from software engineering will not be applicable on PLC because of the different setup, and specific the IEC 61131-3 language. Simulation approaches are already applied in the industry. Also advanced clone detection mechanism is required for these systems to detect when clone and own is used for discipline purposes. It is crucial to pick the right level of automatic testing to embed on the systems since they are complex. The patterns need to be suited for the systems as well. Different simulations, test approaches and analyses need to be investigated to integrate them into the systems since it has a gap between the simulation and faults that may occur. [7]

The article "Evolution of automation production systems" referred to another article called "Automated PLC Software Testing using adapted UML Sequence Diagrams" that wrote about automated PLC software testing. They used the UML sequence diagrams but it can be applied to any IEC 61131-3 programming language as well. They compared automation to automotive. Automotive has very high testing while automation is almost nonexistence. It is usually done in a hurry. According to the article software and system testing is the most important quality check. This is potential in PLC control software that has not been exploited. The prototype built was based on 3S CoDeSys V3 as a plug-in. A system architecture for model-based software testing of PLC software was used. Their prototype showed that the automation development held a high quality. Thanks to the cyclic execution logic that PLC has, the resulting code needs to be enhanced with elements. [8]

2.2 Tools

When it comes to Siemens their products can be used by third-party applications, but they also have their product lines. For testing they have different strategies that all work for different tasks so it is necessary to look at what they can do. Siemens have different solutions but one that is mandatory in their solutions

is a program called PLCSIM Advanced. PLCSIM Advanced is a program that simulates Siemens PLC:s to become virtual controllers.

While using a virtual controller there is no need for the physical device, which enables the possibilities for distance testing on PLC:s that are already on site. The simulated PLC has an application programming interface (API). The API gives possibilities for testing the device since it is allowed for manipulating the written code. The API also enables the PLC to connect to different virtual models of machines [9]. Figure 1 illustrates how the program is built up, as figure 1 shows that the "SimTableApplication" gives the possibility to connect different virtual models of machines.

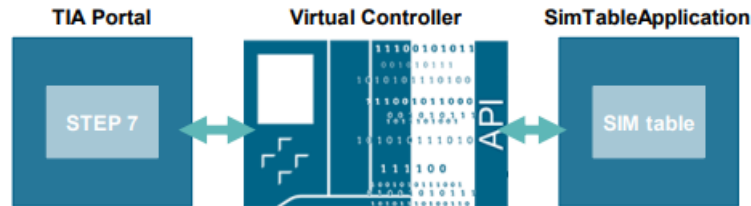


Figure 1: *The figure illustrates how the PLCSIM Advanced works [9].*

The API has an own control panel and the "SimTableApplication" can be anything, it is possible to write an own user program in C++ or C#. It is possible to communicate with the controller through the TIA portal, virtual controller, real Core processing unit (CPU) and Human machine interface (HMI). This is done via ethernet cable and TIA portal on another PC with an ethernet cable between them for connection. The great thing here is that it is possible to modify the time up to 100 times faster than in real testing. This opens up the possibility to run a test in 15 minutes that would take 25 hours to run in real-time and the potential savings is not just in downtime but also in potentially not having to run the machine for 25 hours until seeing an error. The hypothetical savings are huge, but the actual savings won't be that huge since the testing of the software only is half the truth. But if a software is bug-free and working with a well-executed testing base, then it will be easy to update them because the software testing will not be needed to be done on site.

When simulating machines, it is divided into two parts. The first part is to simulate the PLC code, this means that after writing the PLC code it will be simulated on a virtual PLC. The second part of the hardware automatic testing of code is that it is not enough to simulate the controller, there are sensors and so on that will need to be simulated and tested. The controller works with different machines, just because the programmer decided that one variable should be false on the program does not mean that it will work. Because the reality might be the inversed and the programmer thought wrong while writing the tests. But this can be managed, using two different tools from Siemens. S7 Unit test is one of them and the other one is SIMIT. S7 Unit test is the testing tool, and the simulation tool is SIMIT. Less advanced simulation is possible with TIA Portal, the simulation done is done on a PLC and is not as advanced as SIMIT. This type of coding is done through the PLC code and is done in the same environment as the controller since it is written in the same program. It is harder to get an overview of that simulation but is manageable.

2.2.1 S7 Unit test

S7 Unit test is a tool created from Siemens for automatically testing parts of a PLC program. This tool needs the PLCSIM Advanced to be installed and uses that for unit testing. As previously written the PLCSIM Advanced gives a virtual controller but to work with anything it needs an API. S7 Unit test tool is built with two blocks, one is an editor and the second is the application.[10] As figure 2 shows the tool needs to have a TIA Portal project loaded, this means that the PLC code needs to be written for testing.

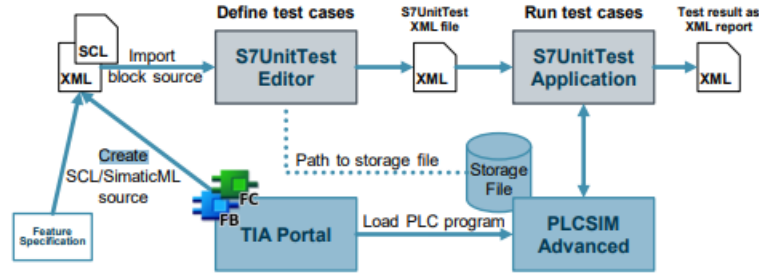


Figure 2: The figure illustrates how the S7 Unit test works.[10]

The editor tool enables the possibility to create and edit a test for a PLC program, the test fixture is a list of cases with individual steps. In these steps, it is possible to manipulate the PLC for errors and when it has run through the application it will log how it handles the tests that were written in the editor and give a result report. As figure 3 shows the test cases are built in an XML File where it is decided what is going to be tested. Then different test cases should be executed and within all those cases some steps will be writing/reading from the virtual PLC. [10]

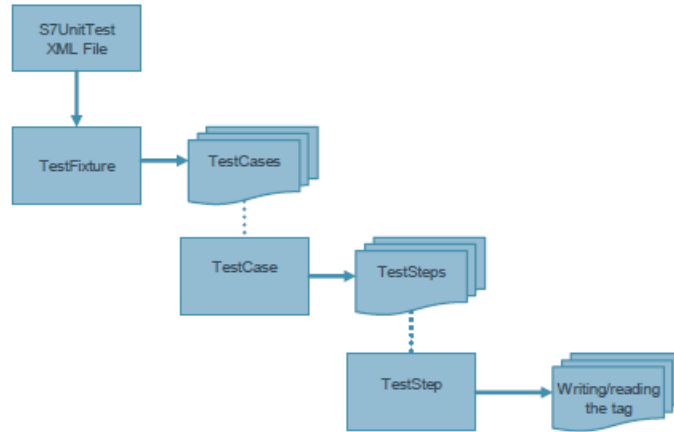


Figure 3: The figure illustrates how the S7 Unit test editor works. [10]

Unit testing is the most common automatic testing in software testing and the most used, this is something that will be run quickly and check the system for faults. The advantages of unit testing are that the testing will be done quickly and the changes within of functions will be tested if they are still working properly. If a unit testing is done properly it will automatically do regression testing with different steps on.

2.2.2 SIMIT

SIMIT is a framework that allows simulations. It contains the following functions:

- Complete plant simulation
- Simulations of signals, devices and plant response
- Input and output simulator of test signals for an automation controller
- Testing and commissioning automation software

These functions are taken from the SIMIT manual. [11] With SIMIT it is possible to work with charts, visualization and coupling. Working with charts is essentially develop a simulation with different components

that are available in the libraries. And working with visualization it shows you an overview of the signals from the plant. Coupling is essentially to specify individually signal to be processed by SIMIT. [11] It is the coupling part that allows SIMIT to work with PLCSIM Advanced and with that it enables the exchange of data between them that simulates the PLC on the virtual plant.

2.3 Model-View-Control

One software design pattern is the Model-view-controller. It is a well-known pattern and commonly used to separate the code, so changes done in the model or view does not affect the rest. The way it works is that the model does the work, the view shows it to you and the controller controls the information to the model. [12] In this thesis the metaphor will be used but also add a dimension that is called testing. The model will be a simulation of the machine, the view will be the HMI and the controller will be the PLC. The testing will be a fourth step in communicating with the PLC. Figure 4 illustrates how the project will be built up in the thesis.

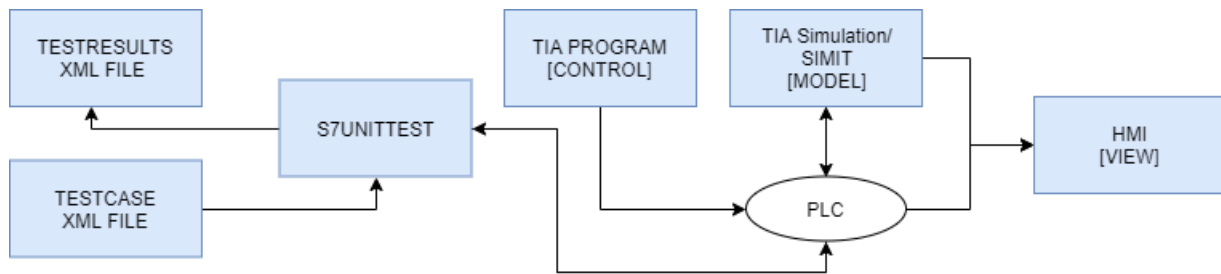


Figure 4: The figure illustrates the modified model-view-controller and how it will be used in this thesis.

This is a common way of handling interfaces and will be used in this case as well. The HMI will be visible for illustrations purposes, even if something simulates it is not interesting to look at the simulation if it is in pure code. That is why a view is available, it is both more interesting and it is easier to look at than code.

2.4 Test automation pyramid

The need for testing has always been known, but the testing code was considered to be expensive to write. [13] One effective strategy is the "Test Automation Pyramid" illustrated in figure 5. The pyramid is usually used in software testing and implemented there, but this strategy could also be implemented in the automation world. All of these can be translated in different ways in the commissioning of a machine. The unit part is the same, the controller is done in code, so there is no strange transition. Service can be translated as the simulation of the controller, or the actual hardware of the machine that is controlled by the PLC code. And the user interface (UI) can be seen as the human machine interface of the machine and the simulation.



Figure 5: The figure illustrates how the method "Test automation pyramid" looks like with a figure.[5] [13]

This would mean that the interpreted test automation pyramid would look like figure 6. The need for automatic testing has always been there, the software has done automatic testing since a while back and the

reason that this has not been done on machines like the software is that the software is only half of it. [13] The other half is the real world, for example there is the need to implement mechatronics. The effort for implementing this has been too expensive to be considered worth it. There are many different strategies on how to distribute the testing, how much should be automated and how much should be manual. According to Nikolova unit testing should be about 90%. [14]

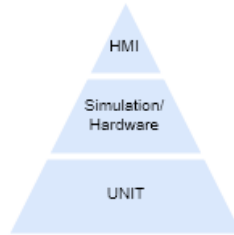


Figure 6: *The figure illustrates how the method "Test automation pyramid" looks like with the changes in a figure.*

But there are companies like Siemens pushing this forward. There are programs like SIMIT as mentioned, and there are several programs in that direction. The tools for implementing the test automation pyramid in the automation exists. It is a matter of implementing those that remain.

3 Machine

To make automatic testing of code possible, the first step is to develop a program to test. When there is a program it is possible to test the code. The platform that will be used for programming is Siemens Tia Portal version 14 and version 15. The testing will be done in the S7 Unit test tool and TIA portal. For this to be done an HMI will be needed to illustrate the TIA portal simulation. To have a code to test a filling machine line program is going to be created. The fill machine line will have theoretical hardware and an actual programming part that is the controller and the simulation. The actual programming part will then be tested in different ways. It will have a part that simulates the controller, one part that is the controller and a view part that is much more interesting to look at. There will also be a simulation done in the controller that will simulate the fill machine line. The actual unit testing is first going to be manual and later automatic with the S7 Unit test on the program created.

3.1 Hardware

PLC used on this project will be a Siemens PLC 1511-1 PN but it could have been any Siemens PLC of the 1500 series that supports PID blocks. The product will be put on a conveyor and will be rolled to different stations. The filling line will have 6 different stations. Figure 7 illustrates the layout of the program and the hardware. For getting the position there will be a laser sensor giving the actual position and with that the controller will keep in a track if there is anything on the line. The conveyor will be run with a motor.

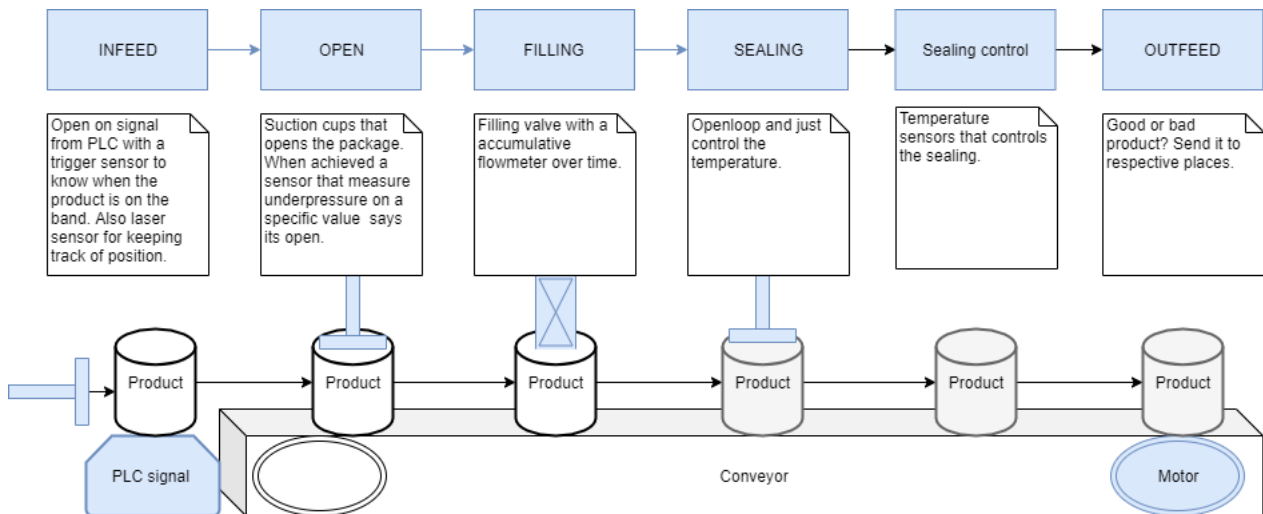


Figure 7: *The layout of the program and hardware.*

The filling line will contain six stations, where all have different tasks and also there is a conveyor that needs to be controlled. The infeed station will open on a signal from the PLC, the product will then be sent to the conveyor. When it is on the conveyor, it will be transported to the photosensor in the open station and stop there. Once it has reached the destination the station will have a job to do. The job is that it needs to open the product as the station's name suggests. There are suction cups that open the package after being sent down by a linear arm. Once the underpressure has reached a value according to a sensor measuring it, the product has successfully opened.

Then the conveyor transports the product to the filling station. Where there is a photosensor telling it has reached the station and valve that will open and close on demand from the controller. It will have a specification of how quickly the maximum flow is reached and how quickly the valve is closed. The sensor will give small pulses of how much fluid is running through, also known as an accumulative flowmeter. The controller will need to keep track of when the product will be full since the sensor only gives pulses.

The next station will be sealing, this will be a burner. In this project, the burner will be an open loop and

just control the temperature outcome of the product that passes. The next step is the sealing control that has temperature sensors that determine if the product is closed or open. The last step is called the outfeed station. Here it is determined where to send the product if it a good product it will be sent on and the conveyor will be empty. But if the product is marked as a bad product, then there is a linear arm shoving it to the side of the conveyor and clearing the band. Since a laser sensor is used it will only be possible to have one product on the band.

3.2 Programmable logic controller programming

The software will have a couple of stations that all have different tasks. When programming a filling line there needs to be a clear functional description. This will make programming the controller much easier. In this case, there will be a maximum of one product on the band. With this requirement, the chosen path of programming will be case statements. The program will need to have a variable that tells the analog output that it wants a product, the easiest way to do this is to use a Boolean variable. In this program, the variable will be called PLCInfeed. And when the laser sensor gives a value that is measured out or specified as zero in the program it will have a product on the band. For this value to be zero there must be a mechanical configuration of the laser sensor. The next step is to control the conveyor.

To control the conveyor we use a motor that rotates the conveyor. One of the hardest things is to move the product and steering the speed of the conveyor band to move to the wished position. The maximum speed for the conveyor band is set through the program. For a filling line to work as expected the product needs to raise the velocity with a constant acceleration and then before arriving it needs to break down slowly for the purpose of the product not falling over. To do this it is needed to use math and physics. To make the program as generic as possible acceleration is given for the program to have a constant acceleration to the maximum speed.

The software has to be built up for real challenges, in the case of the conveyor's speed it has to be ramped for the product not to spill. If the conveyor belt is running at max speed this must be taken into account for in the controller. This creates a problem in programming the conveyor which will be solved by physics. In the coming equations, D is the distance, V the velocity and a is the acceleration. By integrating equation 1 gives equation 2.

$$Velocity = \frac{dD}{dt}, \frac{dV}{dt} = Acceleration \quad (1)$$

$$\int_0^t V(\tau) \cdot d\tau + D(0) = D(t), \int_0^t a(\tau) \cdot d\tau + V(0) = V(t) \quad (2)$$

Then equation 2 becomes 3.

$$S(t) = \frac{1}{2}a \cdot t^2 + V(0) \cdot t + D(0)|_{D(0)=0}, V(t) = a \cdot t + V(0)|_{V(0)=0} \quad (3)$$

If starting values are set to zero as equation 3 shows then equation 4 is gained, which is used in the program. This will work since the acceleration is constant and the start values D(0) and V(0) are zero.

$$Distance = \frac{Velocity^2}{2 * Acceleration} \quad (4)$$

The equation is needed in the controller to know when to start slowing down the speed so that the product will not go too far and need to go back to reach the destination. Before arriving at the photosensor, the product will be running in a crawl walking speed that will make it more accurate and easier to determine that it should stop when the photosensor says it is there. The equation works since the acceleration is constant and the start conditions are zero, that is why the equation is used and they work. Since the speed is low the margin does not need to be high and it will stop in time.

In the controls programming these problems are solved by having a preferred maximal speed and an acceleration. With these variables, it is possible to calculate the distance traveled without having the time. This means that we can calculate where the distance will be and then know when we should decelerate the speed or accelerate up to the max speed. For it to work there are two similar if statements where the program gives a set point for the speed and the acceleration. One of the statements is showed in figure 8. The controller will give set points for the conveyor motor to follow. The reason why it must have the same statement but with reversed signs is that it needs to know when it must start the deceleration. This means that it is needed to use the laser sensor for ramping up the speed. Also, there is a margin of safety added that just means that it will stop earlier than it would, and it will reduce the chance for the product to travel to far. The reason for this is that the product will avoid backing up. While it decelerates it will decelerate to a certain speed and be running in crawl walking speed to stop at the photosensor's response, by doing it like this means that it will not pass the position of the sensors.

```

IF "db".bandVel >= "Simulation_DB".bandVel AND
  (ABS(#movingTo - (#bandVelMax * #bandVelMax / (#bandAccelerationMax * 2.0)) - #marginOfSafety) > "Simulation_DB".movingReal) THEN
  "db".bandAcc := #bandAccelerationMax;
  "db".bandVel := #bandVelMax;
  #hasRamped := false;
END_IF;

```

Figure 8: *The statements that determines the velocity.*

Starting with the station open is getting the product to the sensor. This is done by calling the function that moves the conveyor. Once it has arrived at the photosensors on the open station it will start a timer and opening the product. After the product has arrived the linear arm starts moving in as well as the vacuum suction enables. When it has reached the desired underpressure the suction is turned off and the linear arm starts moving out. When this is done the program sends it to the next case. The controller will mark the product if it is not opened correctly. The timer will handle if the product is stuck, the timer is there only if the product fails to reach the wished underpressure but it must have a maximum time to not slow down the production. This means that the product will be marked as a bad product and not opened and passed on to the next station. Sometimes opening the product will be too much work and marking it as a bad product is the best way to go.

Once again the conveyor function is called for getting the product to the filling stations sensor showing it is there, then if the product is successfully opened it will fill the product. If the product is not opened, then the valve will not open to fill the product since it is marked as a bad product in the controller. But on those cases, they have opened the controller opens the valve and starts accumulating the liquid and once it meets the preferred level it will be passed on to the next station for sealing the product. The accumulative sensor gives pulses of how much liquid is given on and the controller needs to accumulate this. Even here there is a timer set if the product does not fill up for different cases it will be marked as a bad product and passed on to the next station. The filling station has acceleration for the valve, this gives how quick it will reach full open giving a flow velocity. In this part of the code a proportional-integral-derivative (PID) controller will be used for determining when the valve should open and closes. Here the Siemens PID Compact data block will be used with a couple of parameters. It will have a set point, which will be 100 since it is imagined as percent. The input will be the products fill level which is accumulated in the controller by the pulses of the sensor. It will also have the output as a set point flow velocity. Once the product is full it will go to a different state on the case statements. It will also enable the sealer heater, if it has filled the product it will be possible to spill. This means that the heater is started as soon as the fill level is above zero. If the product is empty, then it will just be passed on and not start the heater. If the product were sealed it would mean that it will take up more space when thrown since it would be full of air and unnecessary energy would be wasted.

The sealing station will be an open loop, this means that once the product passes the sealing it will either be on and be given a value to the temperature of the products. The only time it retains value is if the "sealerOn" parameter is enabled in the filling station. The reason for this is that the filling station is keeping track if it is full, or even half full. The sealing heater will also theoretically have more time to get warm for the product to reach the correct temperature.

When arrived at sealing control the controller reads the sensor to see if it has fulfilled the criteria after passing the sealing station. For example, if it still is open then it has not passed the sealing control, but sometimes it is not needed to seal the products for example if they are empty.

The last station is there to control if the product is marked bad product or not. If it is an arm removing it from the band is enabled pushing it off the band and counting the bad products in the software. If the product has fulfilled the set requirements, then it will be passed on as a good product and add one to the product counter. If not, a linear arm will be set to push down the bad product from the band. This will just be a Boolean for the linear arm to push it down from the band.

After completing the cycle there are certain values that require to be restored, this is the first case statement that will always be run. Here all values will be restored in the controller. For example, all timers, the product parameters and so on needs to be restored.

The safety switch will keep track of the last sequence and shut down everything. Once called this will shut down everything, and when it is restored it requires that the product is set as a bad product on the HMI to start running normally again.

3.3 Human machine interface

The HMI is made with the help of the PLC programming, the controller will be simulated with a TIA portal version 15. But they only simulate values and for it to be illustrated in a figure it is needed to have an HMI. To build up the HMI a couple of variables is needed to be embedded in the controller. The distance is already mentioned and explained, but on the HMI, it will be illustrated as the product is moving on the band on the X-axis. Figure 9 shows how the product looks, only the relevant part is shown at the same time with the help of the PLC code. For example, if it has reached open then the open sensor and the linear arm is shown when it has passed the visibility of these will be disabled. And with the help of the simulation, it will be able to move the entities as the arrows show in figure 9. For example, the arm steering if it is going to open will fall to the right position, and then return. The green long posts are sensors showing up when the product has arrived at the correct station. The way chosen to see if the conveyor is rolling is through the circle, it will be dependent on the position and it is green if it is between two stations.

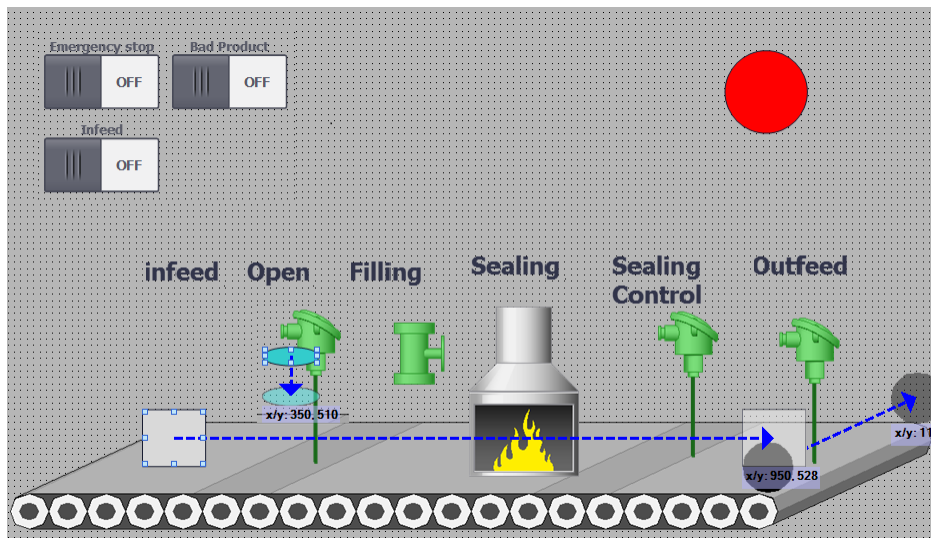


Figure 9: The figure illustrates how the HMI looks like when all states are activated.

Before it reaches the station called "Open" the HMI will look like figure 10. If the program opens the product the outline of the box will be green, if the product is closed then it will still have black outlines. The colors are there to illustrate if it is full, half full or even empty.

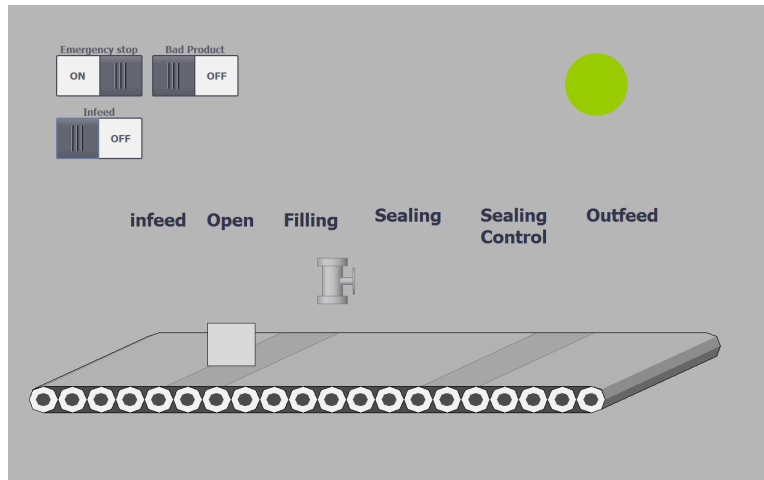


Figure 10: The figure illustrates how the HMI looks like when the band is rolling but has not reached open yet. Also the box is not opened.

For example, the blue circle on the open station and the green sensor is not shown from figure 9 until the product has arrived. The blue circle moves down when the product arrives and the suction power enables until the underpressure has reached the wished value, and then it goes up again before disappearing. The green valve is only green when the valve is open, otherwise it is grey. The sealing grill is only visible when the product needs to be sealed, all these variables are set in the controller's code. The last two stations' sensors are visible when it reaches those destinations. The black circle moves down the band to illustrate a push for the bad products to get pushed down from the band since it is not possible to have both horizontal and diagonal movements on the same entity this was the only solution available.

The box will change colors while different tasks are done if the product is open the outside will be green. If the inside is empty its grey, if it has started filling but is not full the color inside is orange. If it reaches full capacity the product will be dark green separating it from the outside green, so it is visible that it is not sealed. If the product is red, then it is a bad product. The HMI on the TIA portal does not support rolling animations, so the conveyor belt is fixed as well as the Sealing heater and the colors for filling up the product. Figure 11 illustrates how the HMI looks like when the simulation is filling the product.

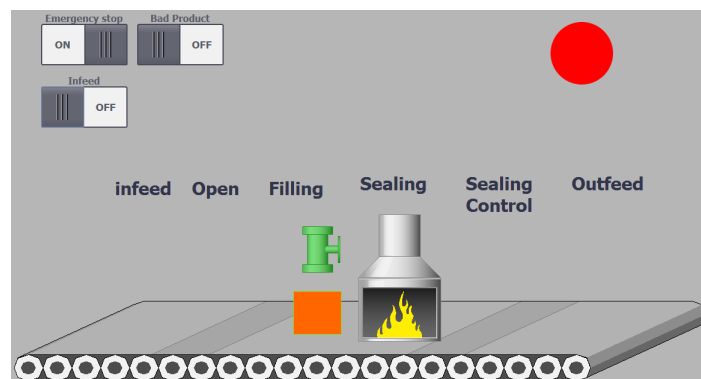


Figure 11: The figure illustrates how the HMI looks like when it is filling up the product. As it is showed the heater is enabled as soon as it starts filling the product.

To illustrate the tests the HMI is kept fairly simple, the movements are linearly and the items are visible or invincible. To make it work properly it must communicate with the PLC where the PLC calculates the range the object will move. The range is set in percent as the rest of the project, from 0-100 percent. But

they are scaled up for a pleasant movement to 10 000 since the move would be stuttering These movements are not as sophisticated as the product is moving. It does not need to keep track of the distance it moves as the product does. Figure 12 shows the black circle, and it will be moving to push down the product from the band.

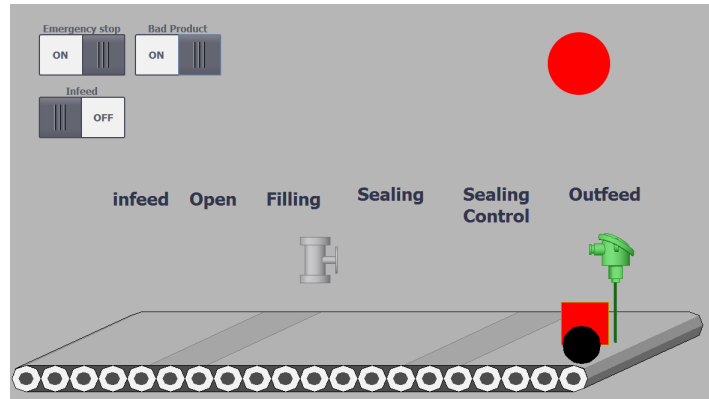


Figure 12: The figure illustrates how the HMI looks like it is filling up the product. As it is showed the heater is enabled as soon as it starts filling the product.

To make the colour scheme less confusing figure 13 shows how they should look like.









Bad product, closed.	Bad product, open.	Empty product, closed.	Empty product, open.	Full product, open.	Full product, closed.	Not empty, open.	Not empty, closed.
Red inside, black outline.	Red inside, green outline.	Grey inside, black outline.	Grey inside, green outline.	Dark green inside, green outline.	Dark green inside, black outline.	Orange inside, green outside.	Orange inside, black outside.
							

Figure 13: The figure illustrates how the HMI looks like when all states are activated.

3.4 Simulation

One kind of automatic test is to simulate the full machine line, for showing how the machine can behave it ease the understanding of the machine. In these examples, all sensors are simulated and the entire machine is fictional.

3.4.1 TIA Portal

The simulation of the machine is done on the controller, but by programming a separate part of the code as a block that runs at a specific cycle time. This enables a simulation possibility since there is an HMI the code can be illustrated. The cyclic time chosen is one millisecond, which means that it will be able to get an accurate simulation. In the simulation, the coding is done by ladder combined with structured text. The ladder has a parameter called simulate, which enables and disables are branches so they will not run when it is not needed.

When simulating a machine like this it is a good idea to use a ladder diagram. This system was chosen since there are many advantages. In figure 14 part of the ladder code is shown. This part is the conveyor part, the ladder is complemented with a structural text code. For example, the Boolean "bandRolling" is used to steer the conveyor, the conveyor is not only dependent on the controller demanding it to stop. It also depends on if the simulated velocity is zero or not. This is due to that the band will not stop immediately and is dependant on the speed and the acceleration. It is also possible to use the ladder in the case of starting sensors that are simplified on ladder. When it reaches different stages is very much dependent on the position of the product, comparing the position with the expected position then turns on the sensors in the simulation.

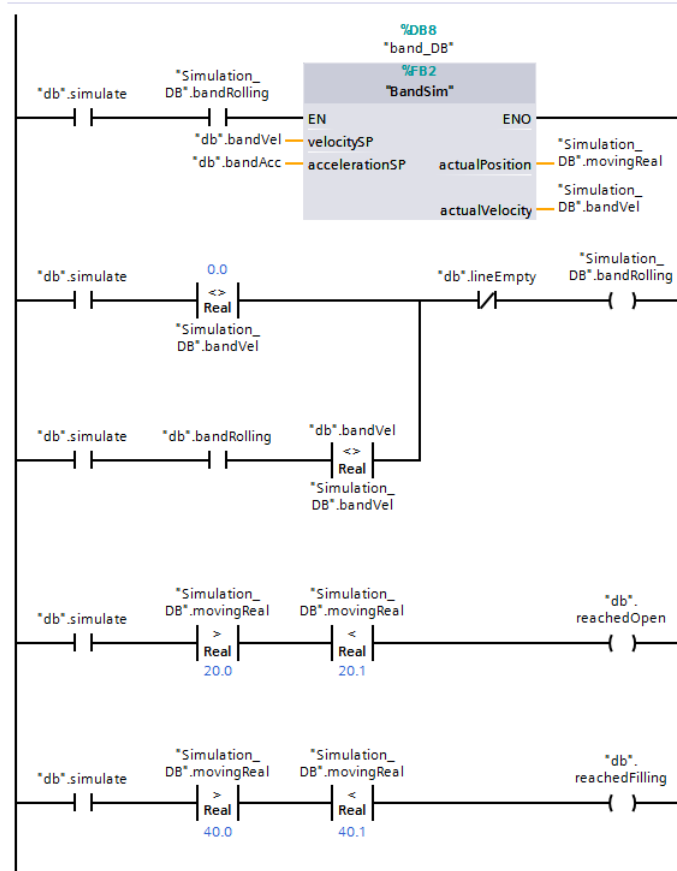


Figure 14: The figure illustrates how the simulation of the conveyor is done on ladder.

There are similarities with the simulation of the filling station versus the conveyor. If the controller asks to open the flow, then the valve starts to open more and more. The dynamics of the valve is going to be linear. To have linear dynamics there needs to be an acceleration and a flow velocity. A valve usually has a data sheet and in the datasheet, it says how the flow chart will behave when it opens. The same is used when closing the valve, therefore it will be a delay in closing the valve and the flow will keep on coming due to that it will not close instantly. Figure 15 illustrates parts of the ladder diagram on how the simulation in the filling valve is done.

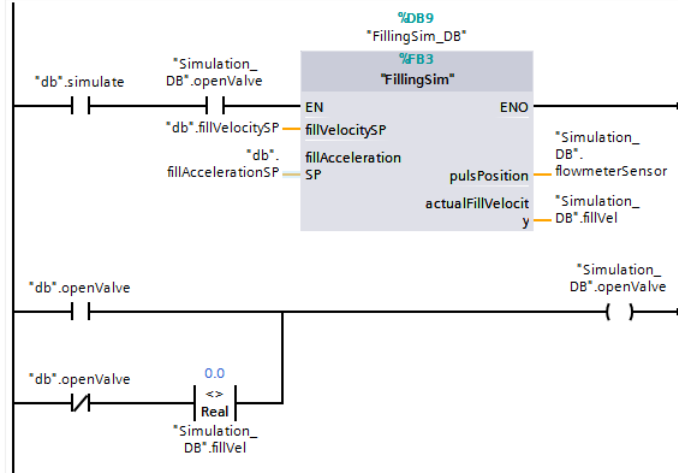


Figure 15: The figure illustrates how the simulation of the filling valve is done on ladder.

The first thing that needs simulating is the infeed. Since the start of the simulation is in ladder it will wait for the user to ask for a product and the line needs to be empty to be run. Then it just sets the value of the laser sensor to zero, which means that there is a product on the band.

Band simulation needs to simulate a position x of the product. Also with this position, it needs to enable the sensors tags, since the simulation is done on ladder combined with structured text it is easy to enable them. In the simulation, the Boolean variable of the sensor will be given true with a width of 0.1. This means that the sensor will be turned on at 20.0, it will be on until 20.1. In this time the conveyor will have to stop. This is the width the product will have on the opening of the product. To get a certain velocity the simulation needs a setpoint speed and an acceleration set point. The simulation will accelerate to the velocity set point and when this is reached it will keep the max velocity until the controller demands a lower velocity with a deceleration value, a negative value. With the acceleration value we use the cyclic time as equation 5 shows, then the value will be a delta velocity which means it is needed to accumulate all $\Delta Velocity$ to get the actual velocity. This is then used as the simulated velocity. When it has reached the maximum velocity, it will keep that value.

$$\Delta Velocity = Acceleration * CyclicTime \quad (5)$$

Getting the distance that the product has moved on the band, equation 6 will be used and then accumulated. It is the same as integrating the velocity over time, which gives the distance. This value is then used for enabling the sensors in the ladder part of the program.

$$\Delta Distance = Velocity * \Delta Time \quad (6)$$

The next step to simulate is the open function, here we only simulate a linear move according to equation 7. This is run when the controller enables the suction and the linear arm is pushed in.

$$\Delta UnderPressure = UnderPressureOld - \frac{CyclicTime}{DesiredTime} \quad (7)$$

Then it is time for filling the product. Just because the controller says the valve should be open does not mean it is going to be open or closed for that part. So, if the flow velocity is not zero means that the valve is still open. The simulation needs to have a tag that is set according to this statement or that the controller wants to open the valve. For simulating this, it is almost the same method used as the band. It needs a set point velocity and an acceleration set point. The acceleration set point is usually taken from the

valve's specification sheet. When this is done it is almost the same as the band velocity, the flow velocity is accumulated as equation 8 shows.

$$FlowVelocity = FlowVelocityOld + FlowAcceleration * CyclicTime \quad (8)$$

The pulse sent from the accumulative sensor is calculated by using the equation 9. Since it is only a pulse the controller needs to accumulate this to know when the product is getting full. This differs from the band since the band has a sensor that will keep track of the position, therefore it needs to give the position in the simulation, while here it is not needed.

$$FlowPuls = FlowVelocity * CyclicTime \quad (9)$$

The sealer only needs to output value if the sealer is on. This is set to 100 in the simulation since it is imagined to be in percent. And the controller doesn't need to be simulated since the sealer is an open loop, which means that the temperature of the product will retain the temperature given in the sealer.

Outfeed is the last step of the stations, if the position is more than 100 and the line is empty according to the controller then the simulation here resets the simulated values of the product position and the linear arms positions.

To make the HMI look fancier there is a couple of simulations, there is one that moves the linear arms in the filling machine line. This is made in ladder blocks, and there is a structural text part. This needs to have a set point for where it should go, in the controller this is set to 100. It is simulated to have a linear movement and reach the set point after two seconds. Also, there is a branch made to scale up the position with 100, so the HMI will be less jumpy. With 100 integers it will be jumpy, so since it is scaled up with 100 it will have 10 000 steps, this means that it will be less jumpy.

3.5 Testing

Testing the fill machine line is the main purpose of the thesis, now there is a controller to test. Testing is divided into two parts; one is a simulation and the other is unit testing. For unit testing Siemens has developed a tool called S7 Unit Test, it is also possible to do unit testing in many ways. It would have been possible to do it in the TIA portal, make a generic function that can be used for programming. Since the persons using it is in a known environment it would have worked. But this is a better approach since it is more user-friendly to make these kinds of tests, and if one person implements the tests and saves them it will be possible to reuse the tests in the future. The best thing about the tests is that if you have 100 test cases and want to run them, they can run and be done in a couple of seconds or minutes depending on the size of the test cases.

For example, say that it is needed to make a change in the program. Instead of standing there and trying to change the parameters by hand or forcing the change in the controller this will save time since it will run all the tests and ease figuring out problems of the code early on development. All this requires that the first person that made the tests made them right. For this to be as good as possible it is good to be more than one that creates the tests, or in the future improves the tests because it is very hard to cover the entire machine by testing it in units. If a machine passes the unit testing, this will not mean that the controller works, it only means that the controller works as it is intended. This is a great way of testing the controller and making changes from a distance while being able to test them. With this program, it is not needed to have a controller, which means that it is not needed to be on site if the controller program is available.

The first thing that is important to understand when using the S7 Unit Test tool is that the steps will be done in three steps.

1. It will write the tag of the input into the simulation at the specified cycle.
2. The program will process that cycle in the simulation.
3. Reads the output tag from the simulation.

These steps are taken from the manual of the S7 Unit test tool. [10] This is crucial to understand before beginning the testing. Also the testing will only read a specific value within a range, say for example the value should not be zero. This is not possible to run in the S7 Unit Test since this function is missing. This flaw is something a tester must keep in mind, but on the other side when testing a controller with a unit test it should be known what to expect.

With an example, if testing a conveyor it has to be known what the controller will demand and test it. The problem that can show up while testing is that if someone in the future changes that variable to something else it will stop working because it has a specific value. It would have been appreciated to have something like the function not in the read parameter as well. In the example above it would have been nice to set that the velocity is not zero if the conveyor is running. This would have enabled change of the PLC code without having to change the test as well. The positive thing is that it is easy to do. Also it would have been nice to be able to rerun the entire test cases more times.

In this project there will be 17 test cases, most of them will test the entire controller rather than focusing on a function. The reason for this is that even if a parameter is changed in one station, it may affect other stations and it would have been impossible to know what has happened after the changes were implemented. If a test runs the entire filling machine, and a change is made in the filling station. This change may affect the sealing control, and if the test is done only on the filling station this will go undetected and would be a problem for another day. The hours spent on this would be much greater than doing it on time. Figure 16 illustrates how the editor looks like. With the editor, test fixtures are created that will have test cases and running them.

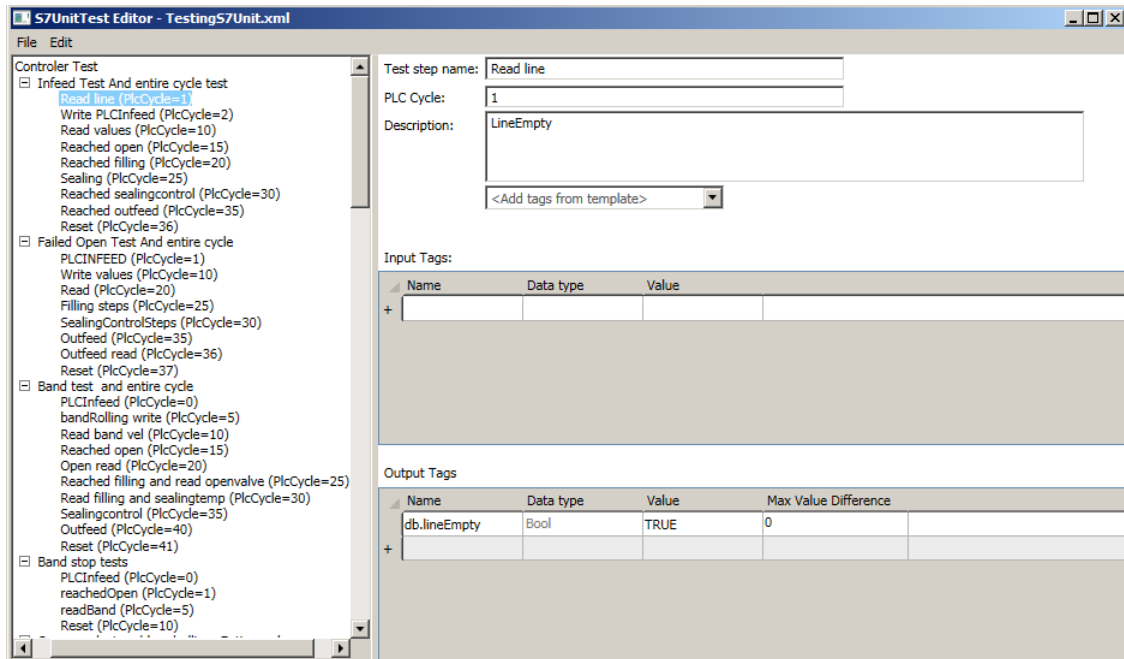


Figure 16: *The S7UnitTest Editor with tests implemented.*

There are two ways of entering tags, first is by importing a block template from the TIA portal program. This is done by selecting the parts of the code desired to test in the product tree, right-click and then generate the source from blocks and the last step is to choose selected blocks only or including all dependent blocks. This way is recommended because it creates an ID-number which will be linked to the program, so if the tag name changes or anything like that the program will automatically detect that change. This only takes the internal data blocks and exports it, it does not consider the entire code and take all of the parameters. For example, if a data block is created, there is no way to export this automatically and keeping the function of ID-number. Also the export does not support its own defined PLC data types, which were used in the

controller. The second way is the way that has been used in this thesis since the controller is built mostly with data blocks and no actual sensors this was the only option.

To create a block template the first step is to open the editor, then choose edit and edit block interface. When this is done a name is set to the block template. Then there is a possibility to choose tag prefix, this means that if it is stored in a data block this will require the data blocks name in front of it. For example a data block is called db and a variable is called lineEmpty, then TIA will ask for ' "db".lineEmpty ' and by specifying this in the prefix it will add db in front of the variables name. Figure 17 illustrates how the program will look after the tags are chosen. There is no ID-number in the XML file created if it is done like this, but since there is no way of importing the data block this was the only option.

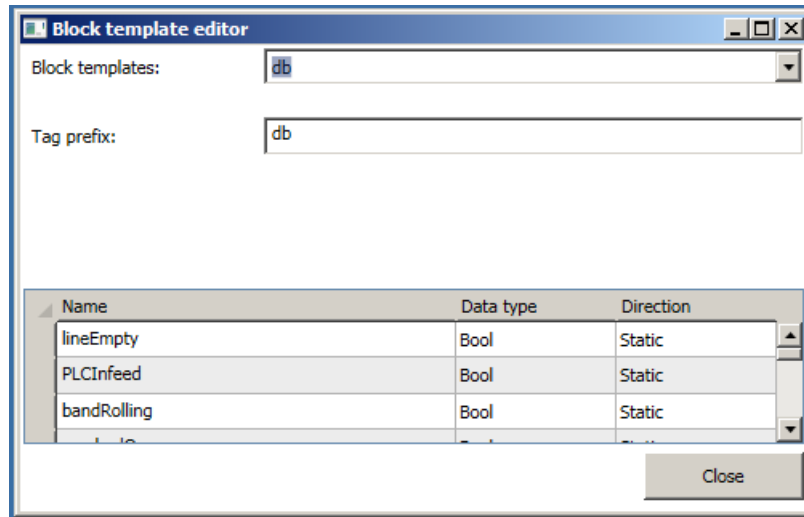


Figure 17: Imported data block called db.

If a variable is changed in the TIA portal program, then the program will not automatically detect this change. That means that every change done needs to be changed in the tests as well. So, changing data types after making the tests is time-consuming.

In a filling machine there are many possibilities for what can go wrong. Testing this project will have 17 different test cases, some are redundant which is completely fine since the program does not make it possible to rerun the test more than once at a time. Also it is needed to write the sensor's expected answer, for example the photosensor at the open station needs to be written true when it arrives.

The testing is made such that it has a foundation, this is being used in all of the tests. In the foundation, it is thought that everything is fine and working correctly. Then the variables wanted to test in certain places replaces the same variables in the foundation. That will mean that the entire cycle will be tested everywhere except in one of the tests. The tests will have different PLC Cycles, which means that they will be run at different speeds. This is done for testing purposes, to test the program and see if the PLC will manage to keep up the speed or if it needs more gaps between reading and writing.

The tests are the following:

1. Infeed test and entire cycle test.
2. Failed open test and entire cycle test.
3. Band test and entire cycle test.
4. Band stop test.
5. Open product, bandRolling and entire cycle test.
6. Started filling? Filling test, entire cycle test.

7. Full? Entire cycle test.
8. Bad product and empty. Entire cycle test.
9. Bad product, half full. Entire cycle test.
10. Bad product, overfull. Entire cycle test.
11. Sealer, entire cycle test.
12. Sealing control, entire cycle test.
13. Failed sealing control, entire cycle test.
14. Sealing control Bad product, entire cycle test.
15. Sealing control Bad product open and sent on after timer went off, entire cycle test.
16. Outfeed Line empty and good product, entire cycle test.
17. Outfeed line empty after bad product and push off band, entire cycle test.

Figure 18 illustrates how the test cases look like with descriptions. This is one case with many steps. Every step has different tasks to do.

Test case name:	Infeed Test And entire cycle test
Author:	Flamur Breznica
Create Date:	2019-12-05 14:53:06
Version:	0.0.1
Description:	Testing Infeed

Name	Plc Cycle	Input Tags	Output Tags	Description
Read line	1	(Collection)	(Collection)	Reading LineEmpty
Write PLCInfeed	2	(Collection)	(Collection)	Writing PLCINFEED
Read values	10	(Collection)	(Collection)	Reading lineEmpty, movingInt, SequenceStates, bandVel, bandAcc and bandRolling.
Reached open	15	(Collection)	(Collection)	Writing reachedOpen, simulation underpressure, movingpushSP.
Reached filling	20	(Collection)	(Collection)	Writing reachedFilling, simulation flowmeterSensor, simulations sealerNewTemperature. Reading openvalve and products fillLevel.
Reached sealingcontrol	30	(Collection)	(Collection)	Writing reachedSealingControl. Reading products sealingTemperature.
Reached outfeed	35	(Collection)	(Collection)	Writing reachedOutfeed and simulations movingPush. Reading SequenceStates.
Reset	36	(Collection)	(Collection)	Restoring.

Figure 18: *Picture of how the infeed test cases look like.*

Figure 19 illustrates how the test cases look like with descriptions. The figure is taken from the foundation and it is on the filling step. Here the tests will be written, executed and then read.

Test step name: Reached filling

PLC Cycle: 20

Description: Writing reachedFilling, simulation flowmeterSensor, simulations sealerNewTemperature. Reading openvalve and products fillLevel.

<Add tags from template>

Input Tags:

Name	Data type	Value
db.reachedFilling	Bool	true
Simulation_DB.flowmeterSensor	Real	100.0
Simulation_DB.sealerNewTemperature	Real	100.0

Output Tags

Name	Data type	Value	Max Value Difference
db.openValve	Bool	true	0
fb_main_DB.Product.fillLevel	Real	100.0	1

Figure 19: Picture of how the infeed test cases looks like.

3.5.1 Loading TIA project

Defining the test fixture is important, the name of the test fixture can be chosen freely. The name of the PLC instance, on the other hand, needs to avoid dashes. With dashes, it will not find the CPU and will not be able to start it. In figure 20 it is shown how the test fixture is defined. It is not needed to define the CPU type, this will be done generic through the TIA portal loaded to the CPU.

S7UnitTest Editor - TestFull.xml

File Edit

- S7Test
 - Infeed Test And entire cycle tes
 - Failed Open Test And entire cyc
 - Band test and entire cycle
 - Band stop tests
 - Open product and bandrolling -f
 - Started filling? Filling test, entire
 - Full?
 - Bad product and empty

Test fixture name: S7Test

Name of PLC Instance: S7Test

PLCSim Adv storage file path: Documents\Siemens\Simatic\Simulation\Runtime\Persistence

CPU Type: CPU1500_Unspecified

Figure 20: Figure of how the test fixture is defined.

The trickiest part was to find the PLCSim Advanced storage file path, this was tricky since it was not possible to use dashes in the name of the PLC instance. Figure 21 illustrates what button shows the PLCSim Advanced storage file path, copy that location and paste it.

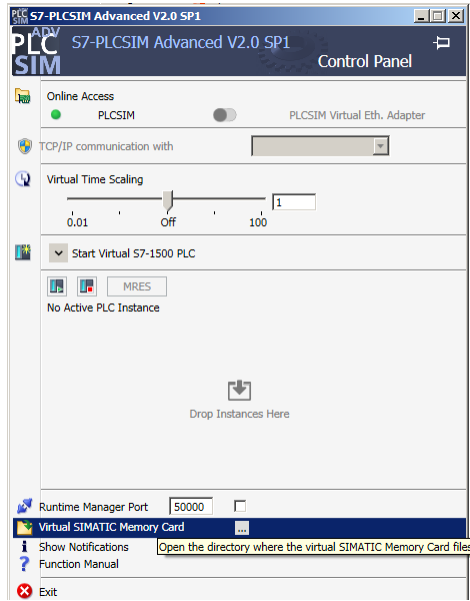


Figure 21: Figure of how the PLCSim Advanced storage file path is found.

To load the TIA project in the simulated PLC it is done through S7 Unit test GUI. First of all, it is necessary to browse the test XML file created previously, once this is done it is powered on through the button that says power on, and when this is done it will be possible to load the program to the right PLC. This is done through choosing extensive online downloads on the TIA portal and it is necessary to search for the PLC and choose it for downloading the project. After this is done it is straight forward, you need to power off the PLC and then press start test. The results will be created in the same file path as the test fixture was located but have the name "_Result" after the name. After every successful run, it will do this, but if it crashes it will not remember the previous result and needs to run again. Figure 22 illustrates how the GUI looks like.

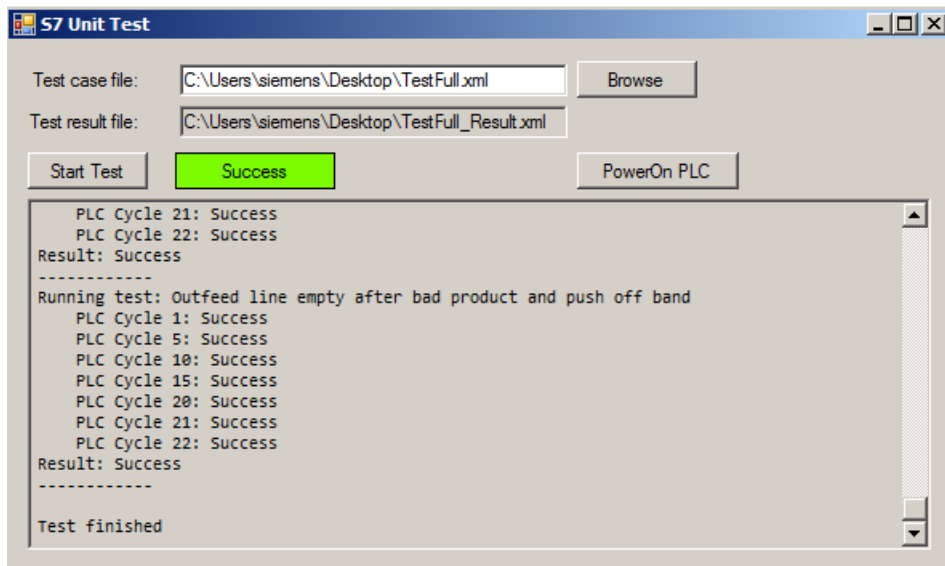


Figure 22: Figure of how the S7 Unit Test GUI looks like after a successful test.

4 Result

Here will the results of the implemented work be written. The results of the simulation and parts of the results of the unit tests will be shown.

4.1 TIA Portal

To get a better view of how the result of the simulation looks like figure 23 shows the position, velocity and the measured acceleration. If the figure is zoomed in the velocity of the band is in crawl walking mode until it has reached the wished position. This could be faster with a PID controller, or if there is no safety margin. This result is simulated and measured, but the controller is used for getting values and telling the simulation of what will happen. Also, here it is needed to reset the pointer to the actual value.

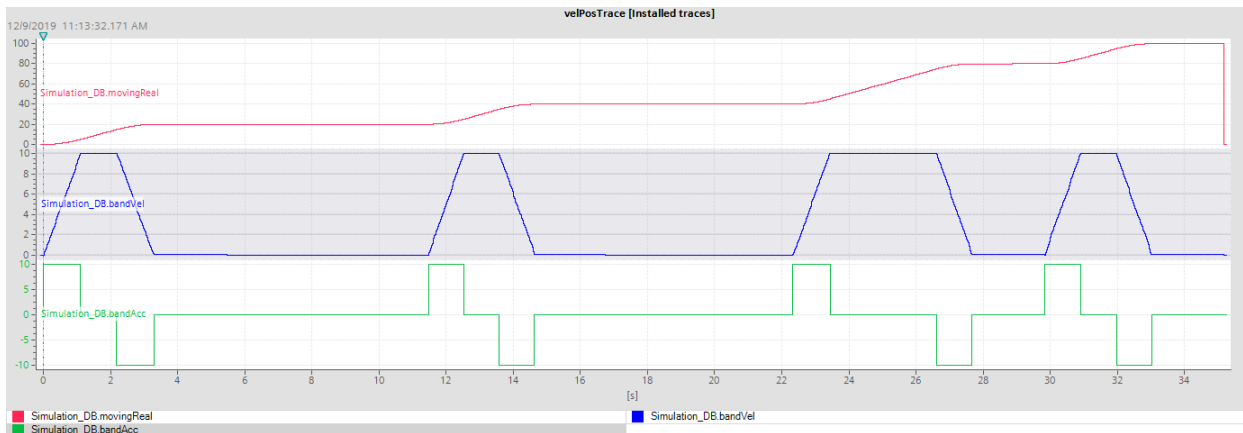


Figure 23: *The simulated position, velocity and acceleration.*

In the simulation the open is simulated as figure 24 illustrates. This makes a linear line with negative inclination which simulates the opening of the product. The controller only looks if it has reached the desired values for it to be set as opened.

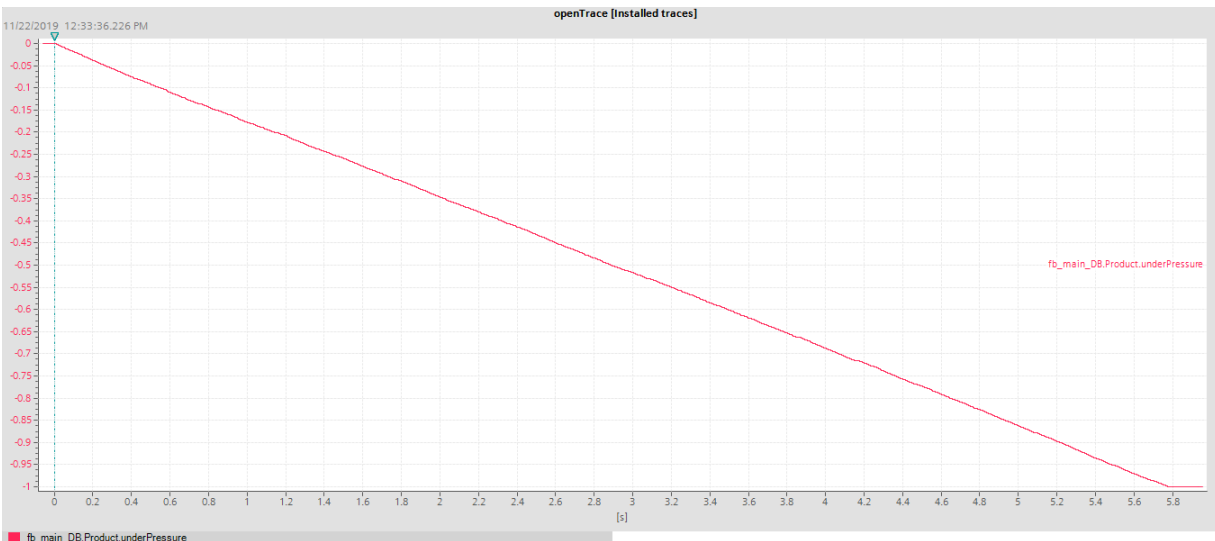


Figure 24: *The chart of the simulated underpressure.*

The result of this is showed in figure 25. This is how the PID controller behaves.

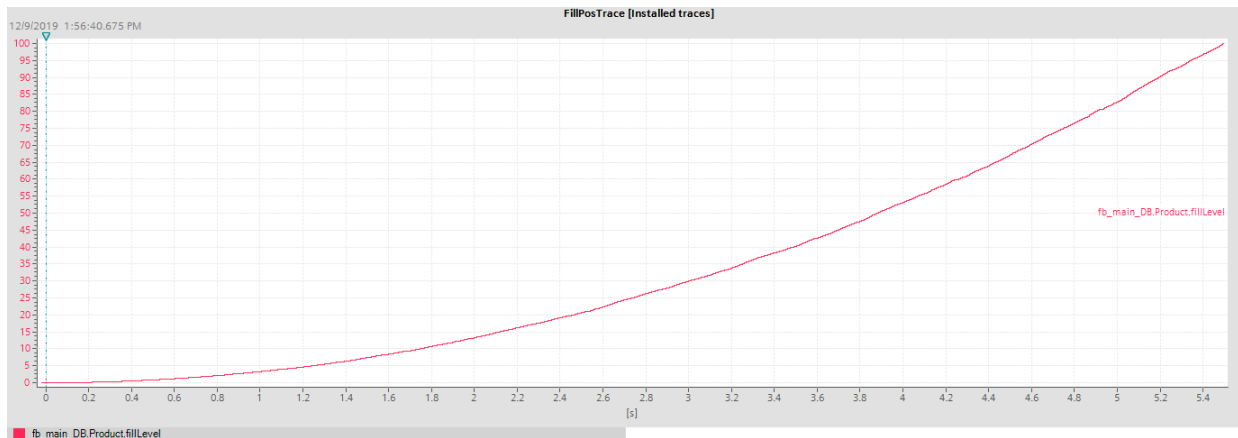


Figure 25: *The accumulated simulated filling of the product.*

Figure 23, 24 and 25 shows different types of charts. The first is speed-dependent and has a ramp start and ramps down the speed again. The second one is linear and the third one uses a PID controller for filling up the product. This is done for learning purposes, it would have been easier to run the PID controller on all three but doing it in different ways is more challenging.

4.2 Unit testing

At the start of the program development, unit testing saves most time according to the report written by Klamme and Ramler. [5] Since this testing method was available late in the thesis it was most problems with the test rather than having a controller that did not work. But even if it was that late, there was a couple of bugs found by the testing method that was not shown earlier on. Even though there were manual testing and simulation there were still errors to correct in the controller's code. For example, when there was a good product, but it was given bad product status it got stuck in the station sealing control because the set statement was inside three if cases. The automatic unit testing came late in the thesis but could be implemented by preparing them, this shows that manual testing still has a disadvantage because it was missed.

The result is given in an XML file, this can be saved as an HTM file as well for opening them everywhere without an XML handler. The result is given mostly with tables and is divided into different rows. The first row is the test result given, in figure 26 we can see that the result of the test was a success. It has three states, they are success, failure and error. The worst of all tests written is shown. It is possible to have a fully functional test and still get an error. So, if it skips a PLC cycle it is possible to run it again and get the success of the tests.

Test result

- **Execution date:** 2019-12-12T15:36:32.5093822Z
- **Result:** Success

Figure 26: *The top of the test result in S7 Unit test.*

The next part of the file is the cases sorted after how they are run through. The figure 27 shows how it looks like. It is divided into different steps and the steps are run at specific cycles. The direction output is what is read from the controller and compared with the expected value if it is the same the result will be a success.

The next cycle is with an input, here the value is written and the result is determined by if it is written or not. The report is long and looks the same. The test result is the same, then it depends on how many cases are written and steps in the cases. They are sorted by the way they are written in the editor.

Test case "**Infeed Test And entire cycle test**"

- **Author:** Flamur Breznica
- **Description:** Testing Infeed
- **Execution date:** 2019-12-12T15:36:32.5133824Z
- **Result:** Success

Test Step (PLC Cycle: 1)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
OUTPUT	db.lineEmpty	Bool	Bool	True	TRUE	Success	

Test Step (PLC Cycle: 2)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.PLCInfeed		Bool		TRUE	Success	

Figure 27: *The part after result in S7 Unit test report.*

Sometimes the S7 Unit test tool gets an error when it should not, then the results look like in figure 28. This means that when the S7 Unit tool wanted to write something on the simulated PLC, it could not reach it. Therefore, it will give an error on the result and the message will be that it failed to run a PLC cycle. When this happens, it will be needed to run the test once again.

Test case "**Sealing control Bad product open and sent on after timer**"

- **Author:** Flamur Breznica
- **Execution date:** 2019-12-12T13:52:49.9287294Z
- **Result:** Error
- **Message:** Failed to run PLC cycle!

Figure 28: *How the result looks like when it failed to run PLC cycle.*

Then there is the case of failure when the value is not like the one expected like figure 29. This means that there is something wrong with the controller since the expected value did not match with the real value.

Test result

- Execution date: 2019-12-16T20:54:54.2087043Z
- Result: Failure
- Message: Value of PLC Tag "db.lineEmpty" doesn't match! Expected: FALSE, But was: True

Test case "Infeed Test And entire cycle test"

- Author: Flamur Breznica
- Description: Testing Infeed
- Execution date: 2019-12-16T20:54:54.2137046Z
- Result: Failure
- Message: Value of PLC Tag "db.lineEmpty" doesn't match! Expected: FALSE, But was: True

Test Step (PLC Cycle: 1)

- Result: Failure
- Message: Value of PLC Tag "db.lineEmpty" doesn't match! Expected: FALSE, But was: True

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
OUTPUT	db.lineEmpty	Bool	Bool	True	FALSE	Failure	Value of PLC Tag "db.lineEmpty" doesn't match! Expected: FALSE, But was: True

Figure 29: How the result looks like when values do not match.

All test cases contain many approximately ten steps each. It needs to run approximately 400 cycles. This means that if the PLC runs at 1ms, all the tests will be run at less than half of a second. Most PLC runs even faster than this and the PLCSIM advanced even allows us to speed up the process to 10 times faster, this means that it will run even ten times the speed than previously suggested. But since the test program used to stop the PLC simulation after every test case it takes much longer. It usually gets an error on starting the tests, in that case all that is needed to do is to start the test again. With this error message, the 400 cycles take about 55 seconds. If this error would not show up, then the time would be reduced to 36 seconds. These times are still low if an engineer would have done this the engineer would need at least a couple of hours to run these tests. And to expect a specific report from the engineer would raise the time even more.

When the testing is done the results are easy to understand. At the top of the results, there is a complete summary and in the summary, you can spot the worst result. There will be a message written as well explaining the problem with the test if there is an error it is necessary to retry the tests because they may be wrong. The most probable error is the failure to write on the PLC cycles which depend on the simulation and not the controller's code.

If the tests are poorly written, then the S7 Unit test may crash. For example if the XML file with test cases misses one type in one variable this will lead to a crash of the program. If this happens it is important to open PLCSIM Advanced and shutdown the PLC since it will be enabled, and this will cause problems with running the tests again.

5 Discussion

When push comes to shove this thesis is about automated testing or at least parts of it. It is not possible to do all the testing of commissioning an entire fill machine with only automated tests. But there is a way of removing parts of the time required and to go from a manual mode in some areas to automatic. By doing this you are not only saving time on commissioning the machine, but it will be usable in the future when an update is needed this might be something that is crucial for the operator on the field.

By making the tests before the operator is out on the field and making a connection between versions and tests, there is a possibility that the prioritized tests will show faults of the update and save time. [4] This is not implemented in the thesis since the machine is not commissioned but is worth taking up and remembering that if there are different versions of the software it would be a great way of connecting the tests to the versions to minimize the time on site. For example if the new version wanted to fix a known fault in the machine, then this test should be prioritized since it is important and it is what differs from the last version. In this way, it is possible to save time on site if it will be needed to test on site. It might be possible and sometimes even cost-saving to simulate the machine and test the simulation before going on site since often when an employed goes out on site it will have higher costs than working in the office.

The report written by Ulewicz and Vogel-Heuser contains a good strategy. [4] The best part about it is that it focuses on making relations of tests and functions. The different versions may have different requirements. And if changes are implemented in the systems it is needed to implement new tests above the old test. The tests are never completed and will have newer versions all the time. In this way, the machines can have the same testing conditions as another machine with an older version. Analyzing the codes will take up time, sometime this may be overdoing it but in other cases this may be the only way to find out flaws. By doing this it is possible to know what needs to be prioritized and tested. [4]

The article written about evolution in automation production systems is written and published five years ago. It takes up the problems and the flaws of automation production systems. In the article, they write about continuous integration which in other words means that it needs to convert the hardware to software. This is done through simulation, and there are tools to do it but they need to be used. There have been several attempts to implement new languages but IEC 61131-3 language remains to be the most used in automation production systems. There are options today to do this, by simulating, unit testing and physical testings on the HMI. There are several ways to do it in the market. For example, since the language is still being used and quite often the manufacturers should try to implement "Static code analysis" on this language. This would help to find errors as soon as they are written. The machines are built for a long lifetime, they should be designed and tested thereafter. According to the article, the testing on the automation production systems is almost non-existence. This is being overlooked due to time pressure. In automotive the tests are about 40-80% of the time, this shows that testing is needed. Instead companies ship of products that are working but not tested and maintenance is hard to do. Some faults are critical and will show after a long time. The article wrote about simulation as well, the simulation is a great way of moving the real world. But the code still needs to be analyzed to figure out these critical errors since the simulation will not detect them. [7]

The test pyramid shows that it is possible to make testing automatic in machines. It is important to interpret the pyramid and translate it for automation. There are tools for doing the pyramid in automation, they are important and probably in the future it will not work without having them. They will also be easier to understand, and maybe an independent program is not needed for testing like it is today. S7 Unit test tool has its interface and needs S7 PLCSIM Advanced to work, maybe in the future it will be more like what JUnit is in java. Embedded in the program code, this would be more comfortable for the programmers since it would not need a change of environment.

The pyramid tends to underestimate the need for GUI testing. It is possible that this can be overlooked in automation and tested on the machine with traditional methods. Implementing automatic testing does not lower the time consumption of testing since if a person is doing this there can be a machine as well. But by doing this it will be needed to write the tests as well, and the writing test takes time. Even when the tests are automated, it will need time and effort to analyze the results. [5] But as Mike Cohn said, it is easier to

have the testing automated and say there is a bug on that line than a tester saying this did not match. [13] In the unit test tool that is used in this thesis, it will not give the exact line but will narrow it down since it runs in cycles. It will be easier to find the fault like this than it would be having a person test the system and say this was wrong.

For example, if there is a value set that is expected and it does not match the test tool, it will write what the read value was instead of telling you that it did not work. This is done quickly and automatically but this creates a need for reading the results. The tests done in the report written by Klamme showed that creating a GUI testing tool should not be neglected. The GUI test they created helped to find several bugs including one in the Oracles code. The flaw with this was that the most faults depended on the GUI programming. Therefore the unit test could not find them. [5] Even if the GUI testing can be overlooked on automation, this shows that all tests will not disappear. Not even on software but it is possible to lower the faults.

Then there is the great thing about unit testing, just because a new function is implemented there is no need of changing the tests. Unit testing done will figure out the regression faults and it is possible to correct them. It gives the possibility to have a great testing tool at all time, this requires that it needs to be kept intact. For example, if a person does a test on site and sees a problem or a unit that is needed on the old unit test version this needs to be updated regularly. By doing this, and if there are many persons involved it will become better and better. The changes made can be described in the unit test tool and the future will help the next update. The tests will improve over time if they are continually updated with new conditions, especially when the old are affected by the new function. There is no wonder that unit testing is so huge in software developments since it tests and gives a straightforward answer. It will narrow down the reason for the error by giving you the read value and then it is possible to compare it with the expected value. By doing this it is possible to find out the reason for the fault, in some cases it will be the unit testing that is poorly written and in some cases, it will be the controller.

The tests are only as good as the person writing the tests, and together humans are better than being alone. Sometimes it will be one person thinking of a test and the next time someone else. It will be the best way of making a great test of a machine, and probably in the future the tests are good enough to send out an update without needing to be on site. By building it brick by brick it will become better and better over time, but it is important to try the tests and examine that the tests work and that it gives the desired result. If it is not tested and examined, then it might be fully working in the future but the tests were wrong from the beginning. This would lead to cost more time and money than not having automatic unit testing. There are advantages and disadvantages to automatic testing.

The automatic testing done here, took 36 seconds to run. Implementing the tests took about 6 hours to write. Doing this manually, they would take about the same time as implementing the tests since the report had to be written as well. This means that the time has been moved, but if it needs to be redone again then it will be better with automatic testing since it will only take 36 seconds. It would be here most of the automatic testing should be done, on unit testing it is worth investing in making tests better. Usually, you move the time but if you need to do it again you save time. According to Nikolova, the cost will rise as you go higher on the pyramid. [14] To save time with automatic testing is all about prioritizing and choose wisely. To automate unit testing is usually profitable and according to Nikolova, 90 percent of the unit testing should be automatic. Creating automatic testing can be very challenging if the object is to save money and time, but the tools are there if the desire to automate tests is. Then it is possible to combine manual testing with automatic in the upper part of the pyramid.

The traditional simulation method was implemented in this thesis. To make this work it was needed to think outside the box and to incorporate both physics and math into the programming. When this was implemented the block used specific cycle times. The cycle time was set in the program and with the cycle time, most of the simulation could be done. For example, if time and acceleration are known then you have both distance and speed according to equation 1 and 3. An advantage of this was that the programmer did not have to learn a new environment. By implementing this as code there were fewer conflicts of interests and separating the simulation code and the controller's code which gave a good overview. A drawback with this way is that the simulation is done in the controller, which means that it takes up both speed and space out of the controller which means it will get limited and not be able to run at maximum speed since it takes up computing power as well. Since this is no real simulation platform it will be up to the programmer

making the simulation as good as possible. But all the tools needed are not there, it will either be needed to implement them or use another platform.

The simulation that was used in this thesis was the traditional way of doing it, the drawbacks with this type of simulation are many. It takes much more time to implement and there are not any finished models to use like there are in different simulation environments. For example in MatLab, there is Simulink that helps with the modules and it is possible to implement real-life physics for this to work. In TIA this corresponds to SIMIT. It has very much in common with Simulink and SIMIT contains many possibilities just as in Simulink. SIMIT also could make a visual content, so it was pictorial in the simulation and illustrative for the person watching it.

If a simulation with a valve is done, it is possible to add noise on the filling, all this is possible to implement with SIMIT. There are many models but it is possible to implement own models using the math operations. It is possible to do the same with the open production, randomize faults and see how the PLC will behave and handle these faults. There also are models that require to have a certain package at a certain price, in other words it is possible to buy models if it is needed. If there is knowledge about how the reality behaves in the area, it is possible to build up models with the math functions in SIMIT. There are also many different finished models that are included. There is a nice way to make visible models as well in SIMIT and a more modern way than the HMI in TIA portal.

In this thesis an HMI was used to make the visualization part of the controller, but this could have been done with SIMIT as well since it offers an upgraded and more modern models. In SIMIT it even was possible to scale the pictures, for example if a product is going to be filled it was possible to scale the filling, so it filled up the product more and more. SIMIT had an example program that was investigated and compared to the way this thesis went with implementing the simulation. The best part about SIMIT is that it has an easier overview of the things going on. When doing math in code it can be hard to grasp, but in SIMIT it is much more comprehensible and easier to grip. With models, it takes less time to get what is going on. For example, if we are integrating a curve we need to comprehend that the summation of those delta variables means that integration is being done. But on SIMIT the integration sign is pictorial and it only needs parameters in like blocks usually does in simulation programs. This is easier to comprehend, and the time needed to create this should be shorter since it is harder to get lost. Another drawback by implementing it in TIA is that it will burden the CPU on the PLC and it will slow down the performance in production. The biggest gain is here, by doing it on a PC and on SIMIT the PC CPU will be heavily loaded. The problem is that it has to simulate the hardware on the PC, so once again it might not be as quick as in reality. But with today's modern CPU:s this is usually not a problem.

In this thesis there were three different types of methods used in the opening of the product, the filling and also the movement of the conveyor. By opening the product it was linear as figure 24 showed. Here a simple open and close operation was used with a linear opening, the result was that it opened at a specific value and the PLC stopped opening the product after it had surpassed that value. This was random and dependent on how the PLC was simulated. The other method was the filling. Here a PID controller was used, this was harder to implement on the controller. But when it did work the controller controlled the valve very well. It was the same as above, it surpassed the limit put but not by much at all. It delivered and filling it like this is much quicker than filling it linearly. The PID controller is also safer, it will be harder to overfill the product and therefore better to use. This is illustrated in figure 25.

The third method used was controlling the movement of the conveyor. There were two real possibilities, one was doing it with a PID controller again. But this was not very practical, so the implementation was done with the second possibility. This was that the velocity of the band was ramped up in the beginning for the product not to tip over on the conveyor, when it reached maximum velocity it should keep ongoing. And then with the acceleration and velocity calculate when to ramp down as earlier explained. Then there was also a safety margin added for the product not to cross the line and need to back up. The last part was done by crawl walking and combined with the sensors knowing when to stop. This is a decent way to implement it, but in reality it would have lost much time since the crawl walking took more time than intended. Figure 23 illustrates how the velocity behaves like. This could be optimized by testing the hardware, but the better solution would still be to use PID control of the velocity of the conveyor. Then it would be safer as well as quicker since it uses three elements of math, proportional, integration and derivative.

If a comparison is made with for example java or C++, those environments have the testing possibility inside the program environment. For java, you have JUnit as mentioned earlier, this is loaded into the program eclipse which is used for programming. This opens multiple options, not only automatic testing of the units. The reason that the JUnit is so popular among programmers is probably that it is integrated into the environment the programmers are used to. It is very easy to use JUnit, java and JUnit have different syntaxes but for a programmer this is nothing new and is not as hard as taking a new program and running it on the side of the environment it was written in.

By implementing the testing into the TIA portal there would be more possibilities for the program to be tested, in the right environment it is easier to make own solutions of the program in a known environment. This is probably one of the main reasons why testing is not as common as it could be in automation production systems. The program developers do not embrace the testing in the code, instead they do other kinds of programs that are external and takes much time to learn. Switching environment creates insecurity on the software to match and show fault that should not be there. But by doing it in the program, it is easier to get hold of the variables and testing would be easier to do. Also much less time would be needed and the report would have the possibility to guide the programmer to the exact line where the fault came up instead of looking it up by themselves. For example with JUnit, the exact line is given. This could be improved in the TIA Portal.

By understanding how the strategy work can be interpreted and present as a possible contender to get a good testing ground that works in automation production systems. By using the test pyramid combined with the software design pattern model-view-controller will end up in a good testing ground that does not rely on real hardware and many faults will be moved up earlier in the chain. This will result in a cheaper and less risk damaging the machines since the machines are simulated. Also there will not be a need to have the physical device hence it will all be simulated on a computer. This is appreciated and minimizing risk is always welcomed in a company. Using these strategies will also have the possibility to get better results on the machine and get better times on the production. It will be easier to strain the limits since there is no risk in the simulation. But just because it works in the simulation does not mean that this will guarantee real life. This is only a way of testing and to see if the theory can be simulated. In real life, there is unexpected stuff, but sometimes they are negligible and the simulation might work. But the risks for breaking machines are usually protected so the theoretical simulation should be tested in real life to see if they can live up to the expectations set from the simulation. As mentioned, integration is a hot topic in automation production systems. [7] This could be directly linked with an other hot topic that essentially is the same thing, digital twin. Digital twin is that a duplicate of the hardware is done on software and therefore it is called digital twin.

As earlier mentioned in the thesis the testing was earlier overlooked, and hopefully that is not the case anymore even in automation production systems software testing should be done. When simulating a machine it can be used for other purposes than testing the machine as well. For example, with SIMIT it is possible to simulate and illustrate the real machine and what it does. This can be a selling argument and prove a potential buyer that something works rather then it does not.

6 Conclusion

Automation usually means that a process will be more effective and give a better result. But it depends on how the process is automated, the automation might be quality control. But in the case of software automated testing as earlier mentioned it will probably save money which in this industry usually means time. [2] [14] Instead of manual testing of the code, the time spent on testing will be moved to implementing automatic testing that does the tests. If this is used only once it might not be cheaper but time might have been moved instead, but this has reusability that manual testing does not. Automatic testing has many advantages, one of them is that they are consistent and easier to rerun than manual testing. Manual testing is that a person runs the tests if they mess up the tests the time spent will be doubled. But if they are automated and something goes wrong it will be just to rerun them and they will be done in a moment with a report. Automatic testing is more reliable.

But there are risks with automated testing as well as with traditional testing, in automated testing, the tests are prepared more and the computer will only do as told. But if it were a person, that person might see flaws with the tests and tell the programmer of the fault. Having a reaction to testing can be both an advantage and a disadvantage. The disadvantage is that the possibility exists that this can delay if the programmer were right and the tester only stirred things up. If a person has misunderstood the testing requirements, then it will take much more time and effort to redo the testing and also rewrite a report. In this thesis, the automated testing was done with unit testing where a report was received. In simulating the machine it will not give a report but it will be visible if it works or not. The conclusion is that the preferable way to do testing is the automated way. The programmer usually knows what to expect and will evolve, then the testing will be better and better.

This thesis has been started up with programming a PLC that controls the filling line. The next part was to simulate this in TIA and testing it with the S7 Unit test tool. The traditional simulation has pros and cons. Some of the pros are that it is the same environment for the programmer, it is possible to do almost everything and reach all variables. The flaws are that it will be hard to simulate the machine with analog inputs and outputs, so a simulation of the finished code is not done. It will be needed to set the inputs and output for the software to be finished.

The layout of the thesis is made from the strategies that are usable for regression testing in automation. The test pyramid and the Model-view-controller is a good way to overview the way testing can be seen on a PLC. The test pyramid can be interpreted on automation production systems and be implemented. But in automation production systems the terms differ to software as earlier explained. This was later on used in the testing and is a great way of using the strategies and they are implementable on automation production systems. The problem is that they are not cheap, the tools usually have a starting cost as well as they need work to be implemented. In other words, the automatic tests cost more than traditional tests. This is the reason the automated tests have been overlooked, but in the future it will be unavoidable since there is a win in quality as well as there is no need to risk and damage the hardware or to sell a product that has been tested too much.

The most important thing to understand is that automated testing will not remove all the traditional testing, even if they are done correctly the reality will always have a say in it. But it will almost definitely lower the error rate, and even have an evolving curve that will get better and better. I believe that manual testing will always be a part of testing, and making testing mostly automatic will generate better quality and will save time in the long run. The toughest part will be to balance the manual testing and the automatic, how much should be tested automatically and how much should be manual.

7 Future work

For future developments of this thesis, there are many possibilities. This thesis has been done to create automatic testing in automation production systems. By making a machine, simulating and also testing it with unit testing this thesis has shown that this is possible. In the future, some areas can be improved.

The first thing is that the controller can be programmed and tested with the unit test from the beginning logging the faults it catches to have statistics. Then when it has passed all the tests and is behaving as expected it can be simulated with SIMIT instead of a simulation with the TIA portal. This gives a better simulation of the reality as earlier explained and also a more modern visualization of the machine. The possibility of making it look better should not be overlooked. As mentioned in the discussion, a better-looking view will be possible to use as a material when selling a project to a client. This might mean income that would not have been there if it were not for the simulation.

With SIMIT it is possible to add complete drive systems and simulate them. It is important to make the software field ready, adding the hardware in the project for it to work. If this is not done, then the signals will not show up. This is a good testing base, simulating the hardware and therefore moving all to software seems to be the way to go. Evolution is heading that way most probably since the cost for a server is lower than the cost for real hardware. It also has a re-usability that hardware does not, for example if a machine has a special kind of sensor. Then it is possible to make a macro and re-use it on other places, and if this needs changes it is easy to do on the macro. The macro makes code look better as well as fulfill a function. A macro will look like a block, it depends on how they are made but it is possible to make them with inputs and outputs for better illustration of what is going on. While making the simulation, it is possible to add timers, drivers, math functions and the list keeps ongoing. Combining this with the S7 Unit Test tool or an improved program of the S7 Unit test tool should be considered since it completes each other on different time spectrum. S7 Unit test tool can be used at the beginning of the creation.

In the future hopefully the automation production system software will be updated with internal testing and be used by the industry much more since changing the environment is probably the major flaw with the programs that exist today. To use something unknown takes up time and creates insecurity but should be seen as an opportunity to evolve.

Hypothetically an optimal simulation would mean that there would be no need for real hardware. If companies master this then they do not need to have a testing machine, the testing machine breaks and is expensive to repair also having to pay the storage, electricity and material needed for testing. This cost will disappear but there will be a new income, the testing machines could be sold. This would need servers, but servers are cheaper than having machines running.

Bibliography

- [1] Siemens,
"Delivery release TIA Portal V15", Entry ID: 109752224, 2017.
Available:
<https://support.industry.siemens.com/cs/document/109752224/delivery-release-tia-portal-v15?dti=0&lc=en-WW> [Accessed: 2020-02-22]
- [2] X Huang, H Chen, J Qian (2013).
Design and Implementation of an Automatic Tool for Software Regression Test Based On STAF. IEEE proceedings of 2013 3rd International Conference on Computer Science and Network Technology.
- [3]] S Ulewicz, B Vogel-Heuser (2016).
System Regression Test Prioritization in Factory Automation. IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society.
- [4] S Ulewicz, B Vogel-Heuser (2018).
Industrially Applicable System Regression Test Prioritization in Production Automation. IEEE Transactions on Automation Science and Engineering (Volume: 15 , Issue: 4 , Oct. 2018).
- [5] C Klamme, R Ramler (2017).
A Journey from Manual Testing to Automated Test Generation in an Industry Project. 2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C).
- [6] S Ulewicz, D Schütz, B Vogel-Heuser (2014).
Software Changes in Factory Automation. Software Changes in Factory Automation. IECON 2014 - 40th Annual Conference of the IEEE Industrial Electronics Society.
- [7] B Vogel-Heuser, A Fay, I Schaefer, M Tichy (2015).
Evolution of software in automated production systems : Challenges and research directions. Journal of Systems and Software vol 110, Pages 54-84, December 2015.
- [8] B Kormann, D Tikhonov, B Vogel-Heuser (2012).
Automated PLC Software Testing using adapted UML Sequence Diagrams. Proceedings of the 14th IFAC Symposium on Information Control Problems in Manufacturing Bucharest, Romania, May 23-25, 2012.
- [9] Siemens,
"Getting started with S7-PLCSIM Advanced and simulation tables", Entry ID: 109759047, 2018.
Available:
<https://support.industry.siemens.com/cs/document/109759047/erste-schritte-mit-s7-plcsim-advanced-un?dti=0&lc=de-WW> [Accessed: 2019-10-22]
- [10] Siemens,
"S7Unit Test", Entry ID: 109746405, 2018.
Available:
<https://support.industry.siemens.com/cs/ww/en/view/109746405> [Accessed: 2019-10-22]
- [11] Siemens,
"SIMIT Simulation Platform (V10.0)", A5E44876196-AA, 2018.
Available:
https://cache.industry.siemens.com/dl/files/317/109759317/att_956837/v1/SIMIT_enUS_en-US.pdf [Accessed: 2019-10-22]
- [12] G Krasner, S Pope,
"A Description of the Model-View-Controller User Interface Paradigm in the

Smalltalk-80 System" 1988.

Available :

https://www.researchgate.net/profile/Stephen_Pope/publication/248825145_A_cookbook_for_using_the_model_-_view_controller_user_interface_paradigm_in_Smalltalk_-_80/links/5436c5f30cf2643ab9888926/A-cookbook-for-using-the-model-view-controller-user-interface-paradigm-in-Smalltalk-80.pdf [Accessed: 2019-12-10]

[13] M Cohn, The Forgotten Layer of the Test Automation Pyramid 2009.

Available :

<https://www.mountangoatsoftware.com/blog/the-forgotten-layer-of-the-test-automation-pyramid> [Accessed: 2019-12-16]

[14] Z Nikolova, Testing Strategies in an Agile Context 2020.

Available :

https://link.springer.com/content/pdf/10.1007%2F978-3-030-29509-7_9.pdf [Accessed: 2019-12-17]

A S7 unit test result

Test result

- **Execution date:** 2019-12-12T15:36:32.5093822Z
- **Result:** Success

Test case "*Infeed Test And entire cycle test*"

- **Author:** Flamur Breznica
- **Description:** Testing Infeed
- **Execution date:** 2019-12-12T15:36:32.5133824Z
- **Result:** Success

Test Step (PLC Cycle: 1)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
OUTPUT	db.lineEmpty	Bool	Bool	True	TRUE	Success	

Test Step (PLC Cycle: 2)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.PLCIInfeed		Bool		TRUE	Success	

Test Step (PLC Cycle: 10)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
OUTPUT	db.lineEmpty	Bool	Bool	False	FALSE	Success	
OUTPUT	db.movingInt	Int	Int	0	0	Success	
OUTPUT	db.SequenceStates	Int	Int	5	5	Success	
OUTPUT	db.bandVel	Real	Real	10	10.0	Success	
OUTPUT	db.bandAcc	Real	Real	10	10.0	Success	
OUTPUT	db.bandRolling	Bool	Bool	True	TRUE	Success	

Test Step (PLC Cycle: 15)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedOpen		Bool		true	Success	
INPUT	Simulation_DB.actualUnderPressure		Real		-1.0	Success	

INPUT	db.movingPushSP		Real		0	Success	
-------	-----------------	--	------	--	---	---------	--

Test Step (PLC Cycle: 20)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedFilling		Bool		true	Success	
OUTPUT	db.openValve	Bool	Bool	True	true	Success	

Test Step (PLC Cycle: 21)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	Simulation_DB.flowmeterSensor		Real		100.0	Success	
INPUT	Simulation_DB.sealerNewTemperature		Real		100.0	Success	
OUTPUT	fb_main_DB.Product.fillLevel	Real	Real	100	100.0	Success	

Test Step (PLC Cycle: 30)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedSealingControl		Bool		true	Success	
OUTPUT	fb_main_DB.Product.sealingTemp	Real	Real	100	100	Success	

Test Step (PLC Cycle: 35)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedOutfeed		Bool		true	Success	
INPUT	Simulation_DB.movingPush		Real		100.0	Success	
OUTPUT	db.SequenceStates	Int	Int	0	0	Success	

Test Step (PLC Cycle: 36)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.SequenceStates		Int		0	Success	

Test case "Failed Open Test And entire cycle"

- **Author:** Flamur Breznica
- **Execution date:** 2019-12-12T15:36:35.6935643Z
- **Result:** Success

Test Step (PLC Cycle: 1)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.PLCInfeed		Bool		True	Success	

Test Step (PLC Cycle: 10)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedOpen		Bool		true	Success	
INPUT	fb_main_DB.Product.underPressure		Real		0	Success	
INPUT	timerBlockFailedOpen.PT		Time		T#0S	Success	

Test Step (PLC Cycle: 11)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	timerBlockFailedOpen.PT		Time		T#0s	Success	
OUTPUT	fb_main_DB.Product.isOpen	Bool	Bool	False	false	Success	
OUTPUT	timerBlockFailedOpen.Q	Bool	Bool	True	true	Success	

Test Step (PLC Cycle: 25)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedFilling		Bool		true	Success	
INPUT	Simulation_DB.sealerNewTemperature		Real		100.0	Success	
OUTPUT	db.openValve	Bool	Bool	False	false	Success	
OUTPUT	fb_main_DB.Product.badProduct	Bool	Bool	True	true	Success	
OUTPUT	db.SequenceStates	Int	Int	20	20	Success	

Test Step (PLC Cycle: 30)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedSealingControl		Bool		true	Success	

Test Step (PLC Cycle: 35)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedOutfeed		Bool		true	Success	
INPUT	Simulation_DB.movingPush		Real		100.0	Success	

Test Step (PLC Cycle: 36)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
OUTPUT	db.SequenceStates	Int	Int	1	1	Success	

Test Step (PLC Cycle: 37)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.SequenceStates		Int		0	Success	

Test case "*Band test and entire cycle*"

- **Author:** Flamur Breznica
- **Execution date:** 2019-12-12T15:36:38.0847011Z
- **Result:** Success

Test Step (PLC Cycle: 2)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.PLCIInfeed		Bool		true	Success	

Test Step (PLC Cycle: 5)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
-----------	------	------	---------------	-------	----------------	--------	---------

INPUT	db.bandRolling		Bool		true	Success	
OUTPUT	db.lineEmpty	Bool	Bool	False	false	Success	

Test Step (PLC Cycle: 10)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
OUTPUT	db.bandVel	Real	Real	10	10.0	Success	
OUTPUT	db.bandAcc	Real	Real	10	10.0	Success	

Test Step (PLC Cycle: 15)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedOpen		Bool		true	Success	
INPUT	Simulation_DB.actualUnderPressure		Real		-1.0	Success	

Test Step (PLC Cycle: 20)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
OUTPUT	fb_main_DB.Product.underPressure	Real	Real	-1	-1.0	Success	
OUTPUT	db.movingPushSP	Real	Real	0	0	Success	

Test Step (PLC Cycle: 25)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedFilling		Bool		true	Success	
OUTPUT	db.openValve	Bool	Bool	True	true	Success	

Test Step (PLC Cycle: 30)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	Simulation_DB.sealerNewTemperature		Real		100.0	Success	
INPUT	Simulation_DB.flowmeterSensor		Real		100.0	Success	
OUTPUT	fb_main_DB.Product.fillLevel	Real	Real	100	100.0	Success	

Test Step (PLC Cycle: 35)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedSealingControl		Bool		true	Success	
OUTPUT	fb_main_DB.Product.sealingTemp	Real	Real	100	100.0	Success	

Test Step (PLC Cycle: 40)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedOutfeed		Bool		True	Success	
INPUT	Simulation_DB.movingPush		Real		100.0	Success	
OUTPUT	db.SequenceStates	Int	Int	0	0	Success	

Test Step (PLC Cycle: 41)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.SequenceStates		Int		0	Success	

Test case "*Band stop tests*"

- **Author:** Flamur Breznica
- **Execution date:** 2019-12-12T15:36:40.7268522Z
- **Result:** Success

Test Step (PLC Cycle: 1)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.PLCIfeed		Bool		true	Success	

Test Step (PLC Cycle: 2)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedOpen		Bool		True	Success	
OUTPUT	db.lineEmpty	Bool	Bool	False	false	Success	

Test Step (PLC Cycle: 5)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
OUTPUT	db.bandRolling	Bool	Bool	False	false	Success	
OUTPUT	db.bandVel	Real	Real	0	0	Success	
OUTPUT	db.bandAcc	Real	Real	0	0	Success	

Test Step (PLC Cycle: 10)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.SequenceStates		Int		0	Success	

Test case "Open product and bandrolling -Entire cycle"

- **Author:** Flamur Breznica
- **Execution date:** 2019-12-12T15:36:41.7219091Z
- **Result:** Success

Test Step (PLC Cycle: 1)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.PLCIfeed		Bool		true	Success	

Test Step (PLC Cycle: 3)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedOpen		Bool		true	Success	
INPUT	Simulation_DB.actualUnderPressure		Real		-1.0	Success	
OUTPUT	db.lineEmpty	Bool	Bool	False	false	Success	

Test Step (PLC Cycle: 5)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
-----------	------	------	---------------	-------	----------------	--------	---------

OUTPUT	fb_main_DB.Product.isOpen	Bool	Bool	True	true	Success	
OUTPUT	db.SequenceStates	Int	Int	10	10	Success	
OUTPUT	db.bandVel	Real	Real	10	10.0	Success	
OUTPUT	db.bandAcc	Real	Real	10	10.0	Success	
OUTPUT	db.bandRolling	Bool	Bool	True	True	Success	

Test Step (PLC Cycle: 10)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedFilling		Bool		True	Success	
OUTPUT	db.openValve	Bool	Bool	True	True	Success	

Test Step (PLC Cycle: 12)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	Simulation_DB.flowmeterSensor		Real		100.0	Success	
INPUT	Simulation_DB.sealerNewTemperature		Real		100.0	Success	

Test Step (PLC Cycle: 15)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedSealingControl		Bool		true	Success	
OUTPUT	fb_main_DB.Product.fillLevel	Real	Real	100	100.0	Success	
OUTPUT	fb_main_DB.Product.sealingTemp	Real	Real	100	100.0	Success	

Test Step (PLC Cycle: 16)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedOutfeed		Bool		true	Success	

Test Step (PLC Cycle: 17)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
OUTPUT	db.SequenceStates	Int	Int	1	1	Success	

Test Step (PLC Cycle: 18)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.SequenceStates		Int		0	Success	

Test case "Started filling? Filling test, entire cycle"

- **Author:** Flamur Breznica
- **Execution date:** 2019-12-12T15:36:43.1439905Z
- **Result:** Success

Test Step (PLC Cycle: 2)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.PLCIfeed		Bool		true	Success	

Test Step (PLC Cycle: 5)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedOpen		Bool		True	Success	
INPUT	Simulation_DB.actualUnderPressure		Real		-1.0	Success	
OUTPUT	db.lineEmpty	Bool	Bool	False	false	Success	

Test Step (PLC Cycle: 6)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedFilling		Bool		True	Success	
OUTPUT	fb_main_DB.Product.underPressure	Real	Real	-1	-1.0	Success	
OUTPUT	fb_main_DB.Product.isOpen	Bool	Bool	True	TRUE	Success	

Test Step (PLC Cycle: 10)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
OUTPUT	db.openValve	Bool	Bool	True	true	Success	
OUTPUT	db.fillAccelerationSP	Real	Real	10	10.0	Success	

OUTPUT	db.fillVelocitySP	Real	Real	100	100.0	Success	
--------	-------------------	------	------	-----	-------	---------	--

Test Step (PLC Cycle: 15)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	Simulation_DB.flowmeterSensor		Real		100.0	Success	
INPUT	Simulation_DB.sealerNewTemperature		Real		100.0	Success	

Test Step (PLC Cycle: 20)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedSealingControl		Bool		true	Success	
OUTPUT	fb_main_DB.Product.fillLevel	Real	Real	100	100.0	Success	
OUTPUT	fb_main_DB.Product.sealingTemp	Real	Real	100	100.0	Success	

Test Step (PLC Cycle: 25)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedOutfeed		Bool		true	Success	

Test Step (PLC Cycle: 26)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
OUTPUT	db.SequenceStates	Int	Int	1	1	Success	

Test Step (PLC Cycle: 27)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.SequenceStates		Int		0	Success	

Test case "*Full?*"

- **Author:** Flamur Breznica
- **Execution date:** 2019-12-12T15:36:45.0891017Z
- **Result:** Success

Test Step (PLC Cycle: 2)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.PLCInfeed		Bool		true	Success	

Test Step (PLC Cycle: 5)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedOpen		Bool		true	Success	
INPUT	Simulation_DB.actualUnderPressure		Real		-1.0	Success	
OUTPUT	db.lineEmpty	Bool	Bool	False	false	Success	

Test Step (PLC Cycle: 10)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedFilling		Bool		true	Success	

Test Step (PLC Cycle: 15)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
OUTPUT	db.openValve	Bool	Bool	True	true	Success	

Test Step (PLC Cycle: 16)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	Simulation_DB.flowmeterSensor		Real		100.0	Success	
INPUT	Simulation_DB.sealerNewTemperature		Real		100.0	Success	

Test Step (PLC Cycle: 17)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
OUTPUT	fb_main_DB.Product.fillLevel	Real	Real	100	100.0	Success	

OUTPUT	db.openValve	Bool	Bool	False	false	Success	
OUTPUT	db.SequenceStates	Int	Int	20	20	Success	

Test Step (PLC Cycle: 18)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedSealingControl		Bool		true	Success	
OUTPUT	fb_main_DB.Product.sealingTemp	Real	Real	100	100.0	Success	
OUTPUT	fb_main_DB.Product.isOpen	Bool	Bool	False	FALSE	Success	

Test Step (PLC Cycle: 20)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedOutfeed		Bool		True	Success	

Test Step (PLC Cycle: 25)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
OUTPUT	db.SequenceStates	Int	Int	1	1	Success	

Test Step (PLC Cycle: 26)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.SequenceStates		Int		0	Success	

Test case "*Bad product and empty*"

- **Author:** Flamur Breznica
- **Execution date:** 2019-12-12T15:36:46.9082058Z
- **Result:** Success

Test Step (PLC Cycle: 2)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.PLCIInfeed		Bool		true	Success	

Test Step (PLC Cycle: 3)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedOpen		Bool		true	Success	
INPUT	Simulation_DB.actualUnderPressure		Real		-1.0	Success	
OUTPUT	db.lineEmpty	Bool	Bool	False	false	Success	

Test Step (PLC Cycle: 5)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedFilling		Bool		true	Success	

Test Step (PLC Cycle: 10)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	timerblockFailedFilling.PT		Time		T#0s	Success	
OUTPUT	fb_main_DB.Product.badProduct	Bool	Bool	True	true	Success	
OUTPUT	db.sealerOn	Bool	Bool	False	false	Success	
OUTPUT	timerblockFailedFilling.Q	Bool	Bool	True	true	Success	

Test Step (PLC Cycle: 11)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedSealingControl		Bool		True	Success	
OUTPUT	fb_main_DB.Product.sealingTemp	Real	Real	0	0	Success	

Test Step (PLC Cycle: 12)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedOutfeed		Bool		true	Success	
INPUT	Simulation_DB.movingPush		Real		100.0	Success	

Test Step (PLC Cycle: 15)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
OUTPUT	db.SequenceStates	Int	Int	1	1	Success	

Test Step (PLC Cycle: 16)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.SequenceStates		Int		0	Success	

Test case "*Bad product, half full - Entire cycle*"

- **Author:** Flamur Breznica
- **Execution date:** 2019-12-12T15:36:48.258283Z
- **Result:** Success

Test Step (PLC Cycle: 2)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.PLCIInfeed		Bool		true	Success	

Test Step (PLC Cycle: 5)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedOpen		Bool		true	Success	
INPUT	Simulation_DB.actualUnderPressure		Real		-1.0	Success	
OUTPUT	db.lineEmpty	Bool	Bool	False	false	Success	

Test Step (PLC Cycle: 10)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedFilling		Bool		true	Success	
OUTPUT	fb_main_DB.Product.underPressure	Real	Real	-1	-1.0	Success	

Test Step (PLC Cycle: 15)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	Simulation_DB.flowmeterSensor		Real		50.0	Success	
INPUT	timerblockFailedFilling.PT		Time		T#0s	Success	
INPUT	Simulation_DB.sealerNewTemperature		Real		100.0	Success	
OUTPUT	db.openValve	Bool	Bool	False	false	Success	

Test Step (PLC Cycle: 20)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
OUTPUT	fb_main_DB.Product.fillLevel	Real	Real	50	50.0	Success	
OUTPUT	fb_main_DB.Product.badProduct	Bool	Bool	True	true	Success	
OUTPUT	db.sealerOn	Bool	Bool	True	true	Success	
OUTPUT	db.SequenceStates	Int	Int	20	20	Success	
OUTPUT	timerblockFailedFilling.Q	Bool	Bool	True	true	Success	

Test Step (PLC Cycle: 25)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedSealingControl		Bool		true	Success	
OUTPUT	fb_main_DB.Product.sealingTemp	Real	Real	100	100.0	Success	

Test Step (PLC Cycle: 30)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedOutfeed		Bool		true	Success	
INPUT	Simulation_DB.movingPush		Real		100.0	Success	

Test Step (PLC Cycle: 35)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
OUTPUT	db.SequenceStates	Int	Int	1	1	Success	
OUTPUT	db.lineEmpty	Bool	Bool	True	true	Success	

Test Step (PLC Cycle: 36)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.SequenceStates		Int		0	Success	

Test case "*Bad product, overfull.*"

- **Author:** Flamur Breznica
- **Execution date:** 2019-12-12T15:36:50.7314244Z
- **Result:** Success

Test Step (PLC Cycle: 2)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.PLCIfeed		Bool		true	Success	

Test Step (PLC Cycle: 5)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedOpen		Bool		true	Success	
INPUT	Simulation_DB.actualUnderPressure		Real		-1.0	Success	
OUTPUT	db.lineEmpty	Bool	Bool	False	false	Success	

Test Step (PLC Cycle: 7)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedFilling		Bool		true	Success	
OUTPUT	db.openValve	Bool	Bool	True	true	Success	

Test Step (PLC Cycle: 10)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	Simulation_DB.flowmeterSensor		Real		200.0	Success	
INPUT	timerblockFailedFilling.Q		Bool		True	Success	
INPUT	Simulation_DB.sealerNewTemperature		Real		100.0	Success	

OUTPUT	fb_main_DB.Product.fillLevel	Real	Real	200	200.0	Success	
OUTPUT	db.sealerOn	Bool	Bool	True	true	Success	
OUTPUT	fb_main_DB.Product.badProduct	Bool	Bool	True	true	Success	

Test Step (PLC Cycle: 15)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedSealingControl		Bool		true	Success	
OUTPUT	db.SequenceStates	Int	Int	25	25	Success	
OUTPUT	fb_main_DB.Product.sealingTemp	Real	Real	100	100.0	Success	

Test Step (PLC Cycle: 16)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedOutfeed		Bool		true	Success	
INPUT	Simulation_DB.movingPush		Real		100.0	Success	
OUTPUT	db.movingPushSP	Real	Real	100	100.0	Success	
OUTPUT	db.SequenceStates	Int	Int	0	0	Success	

Test Step (PLC Cycle: 17)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.SequenceStates		Int		0	Success	

Test case "Sealer"

- **Author:** Flamur Breznica
- **Execution date:** 2019-12-12T15:36:52.1705067Z
- **Result:** Success

Test Step (PLC Cycle: 1)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.PLCIInfeed		Bool		true	Success	

Test Step (PLC Cycle: 2)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedOpen		Bool		true	Success	
INPUT	Simulation_DB.actualUnderPressure		Real		-1.0	Success	
OUTPUT	db.lineEmpty	Bool	Bool	False	false	Success	

Test Step (PLC Cycle: 3)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedFilling		Bool		true	Success	

Test Step (PLC Cycle: 4)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
OUTPUT	db.openValve	Bool	Bool	True	true	Success	
OUTPUT	fb_main_DB.Product.underPressure	Real	Real	-1	-1.0	Success	

Test Step (PLC Cycle: 5)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	Simulation_DB.flowmeterSensor		Real		100.0	Success	
INPUT	Simulation_DB.sealerNewTemperature		Real		100.0	Success	
OUTPUT	fb_main_DB.Product.fillLevel	Real	Real	100	100.0	Success	
OUTPUT	db.sealerOn	Bool	Bool	True	true	Success	

Test Step (PLC Cycle: 6)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
OUTPUT	fb_main_DB.Product.sealingTemp	Real	Real	100	100	Success	
OUTPUT	db.SequenceStates	Int	Int	20	20	Success	

Test Step (PLC Cycle: 10)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedSealingControl		Bool		true	Success	
OUTPUT	fb_main_DB.Product.sealingTemp	Real	Real	100	100	Success	

Test Step (PLC Cycle: 15)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedOutfeed		Bool		true	Success	
OUTPUT	db.SequenceStates	Int	Int	0	0	Success	

Test Step (PLC Cycle: 16)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.SequenceStates		Int		0	Success	

Test case "Sealing control"

- **Author:** Flamur Breznica
- **Execution date:** 2019-12-12T15:36:53.4905822Z
- **Result:** Success

Test Step (PLC Cycle: 1)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.PLCInfeed		Bool		true	Success	

Test Step (PLC Cycle: 2)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedOpen		Bool		true	Success	
INPUT	Simulation_DB.actualUnderPressure		Real		-1.0	Success	
OUTPUT	db.lineEmpty	Bool	Bool	False	false	Success	

Test Step (PLC Cycle: 3)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedFilling		Bool		true	Success	
INPUT	Simulation_DB.flowmeterSensor		Real		100.0	Success	
INPUT	Simulation_DB.sealerNewTemperature		Real		100.0	Success	

Test Step (PLC Cycle: 5)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedSealingControl		Bool		true	Success	
OUTPUT	fb_main_DB.Product.sealingTemp	Real	Real	100	100.0	Success	
OUTPUT	fb_main_DB.Product.fillLevel	Real	Real	100	100.0	Success	

Test Step (PLC Cycle: 8)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedOutfeed		Bool		true	Success	
OUTPUT	db.sealerOn	Bool	Bool	False	false	Success	
OUTPUT	fb_main_DB.Product.isOpen	Bool	Bool	False	false	Success	

Test Step (PLC Cycle: 9)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
OUTPUT	db.SequenceStates	Int	Int	1	1	Success	
OUTPUT	db.lineEmpty	Bool	Bool	True	true	Success	

Test Step (PLC Cycle: 10)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.SequenceStates		Int		0	Success	

Test case "*Failed sealing control*"

- **Author:** Flamur Breznica
- **Execution date:** 2019-12-12T15:36:54.4846391Z
- **Result:** Success

Test Step (PLC Cycle: 1)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.PLCIInfeed		Bool		true	Success	

Test Step (PLC Cycle: 2)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedOpen		Bool		true	Success	
INPUT	Simulation_DB.actualUnderPressure		Real		-1.0	Success	
OUTPUT	db.lineEmpty	Bool	Bool	False	false	Success	

Test Step (PLC Cycle: 3)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedFilling		Bool		true	Success	
INPUT	Simulation_DB.flowmeterSensor		Real		100.0	Success	
INPUT	Simulation_DB.sealerNewTemperature		Real		99.0	Success	

Test Step (PLC Cycle: 5)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedSealingControl		Bool		true	Success	

Test Step (PLC Cycle: 8)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	timerBlockFailedSealingControl.PT		Time		T#0s	Success	
OUTPUT	fb_main_DB.Product.badProduct	Bool	Bool	True	true	Success	
OUTPUT	db.SequenceStates	Int	Int	25	25	Success	
OUTPUT	fb_main_DB.Product.fillLevel	Real	Real	100	100	Success	

Test Step (PLC Cycle: 9)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedOutfeed		Bool		true	Success	
INPUT	Simulation_DB.movingPush		Real		100	Success	
OUTPUT	timerBlockFailedSealingControl.Q	Bool	Bool	True	true	Success	

Test Step (PLC Cycle: 10)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
OUTPUT	db.SequenceStates	Int	Int	1	1	Success	
OUTPUT	db.lineEmpty	Bool	Bool	True	true	Success	

Test Step (PLC Cycle: 11)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.SequenceStates		Int		0	Success	

Test case "*Sealing control Bad product*"

- **Author:** Flamur Breznica
- **Execution date:** 2019-12-12T15:36:55.6007029Z
- **Result:** Success

Test Step (PLC Cycle: 1)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.PLCIInfeed		Bool		true	Success	

Test Step (PLC Cycle: 2)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedOpen		Bool		true	Success	
INPUT	Simulation_DB.actualUnderPressure		Real		-1.0	Success	
OUTPUT	db.lineEmpty	Bool	Bool	False	false	Success	

Test Step (PLC Cycle: 3)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedFilling		Bool		true	Success	
INPUT	Simulation_DB.flowmeterSensor		Real		100.0	Success	
INPUT	Simulation_DB.sealerNewTemperature		Real		50.0	Success	
OUTPUT	Simulation_DB.actualUnderPressure	Real	Real	-1	-1.0	Success	

Test Step (PLC Cycle: 5)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedSealingControl		Bool		true	Success	
OUTPUT	fb_main_DB.Product.sealingTemp	Real	Real	50	50.0	Success	
OUTPUT	db.SequenceStates	Int	Int	20	20	Success	
OUTPUT	fb_main_DB.Product.fillLevel	Real	Real	100	100	Success	

Test Step (PLC Cycle: 8)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedOutfeed		Bool		true	Success	
INPUT	Simulation_DB.movingPush		Real		100	Success	
INPUT	timerBlockFailedSealingControl.PT		Time		T#0S	Success	
OUTPUT	fb_main_DB.Product.badProduct	Bool	Bool	True	true	Success	

Test Step (PLC Cycle: 9)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
OUTPUT	db.SequenceStates	Int	Int	0	0	Success	
OUTPUT	db.lineEmpty	Bool	Bool	True	true	Success	
OUTPUT	timerBlockFailedSealingControl.Q	Bool	Bool	True	true	Success	

Test Step (PLC Cycle: 10)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.SequenceStates		Int		0	Success	

Test case "Sealing control Bad product open and sent on after timer"

- Author: Flamur Breznica
- Execution date: 2019-12-12T15:36:56.6817648Z
- Result: Success

Test Step (PLC Cycle: 1)

- Result: Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.PLCInfeed		Bool		true	Success	

Test Step (PLC Cycle: 3)

- Result: Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedOpen		Bool		true	Success	
INPUT	Simulation_DB.actualUnderPressure		Real		-1.0	Success	
OUTPUT	db.lineEmpty	Bool	Bool	False	false	Success	

Test Step (PLC Cycle: 5)

- Result: Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedFilling		Bool		true	Success	
INPUT	Simulation_DB.flowmeterSensor		Real		100.0	Success	
INPUT	Simulation_DB.sealerNewTemperature		Real		50.0	Success	
OUTPUT	fb_main_DB.Product.underPressure	Real	Real	-1	-1.0	Success	

Test Step (PLC Cycle: 10)

- Result: Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedSealingControl		Bool		true	Success	

Test Step (PLC Cycle: 11)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	timerBlockFailedSealingControl.PT		Time		T#0S	Success	
OUTPUT	fb_main_DB.Product.isOpen	Bool	Bool	True	true	Success	
OUTPUT	db.colorRange	Int	Int	400	400	Success	
OUTPUT	fb_main_DB.Product.badProduct	Bool	Bool	True	true	Success	
OUTPUT	fb_main_DB.Product.sealingTemp	Real	Real	50	50	Success	
OUTPUT	fb_main_DB.Product.fillLevel	Real	Real	100	100	Success	

Test Step (PLC Cycle: 15)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedOutfeed		Bool		true	Success	
INPUT	Simulation_DB.movingPush		Real		100.0	Success	

Test Step (PLC Cycle: 16)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
OUTPUT	db.SequenceStates	Int	Int	1	1	Success	

Test Step (PLC Cycle: 17)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.SequenceStates		Int		0	Success	

Test case "*Outfeed Line empty and good product*"

- **Author:** Flamur Breznica
- **Execution date:** 2019-12-12T15:36:58.0708442Z
- **Result:** Success

Test Step (PLC Cycle: 2)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
-----------	------	------	---------------	-------	----------------	--------	---------

INPUT	db.PLCInfeed		Bool		true	Success	
-------	--------------	--	------	--	------	---------	--

Test Step (PLC Cycle: 5)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedOpen		Bool		true	Success	
INPUT	Simulation_DB.actualUnderPressure		Real		-1.0	Success	
OUTPUT	db.lineEmpty	Bool	Bool	False	false	Success	

Test Step (PLC Cycle: 10)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedFilling		Bool		true	Success	
INPUT	Simulation_DB.flowmeterSensor		Real		100.0	Success	
INPUT	Simulation_DB.sealerNewTemperature		Real		100.0	Success	
OUTPUT	fb_main_DB.Product.underPressure	Real	Real	-1	-1.0	Success	

Test Step (PLC Cycle: 15)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedSealingControl		Bool		true	Success	
OUTPUT	fb_main_DB.Product.sealingTemp	Real	Real	100	100.0	Success	
OUTPUT	fb_main_DB.Product.fillLevel	Real	Real	100	100.0	Success	

Test Step (PLC Cycle: 20)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedOutfeed		Bool		true	Success	

Test Step (PLC Cycle: 21)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
OUTPUT	db.lineEmpty	Bool	Bool	True	true	Success	

OUTPUT	db.SequenceStates	Int	Int	1	1	Success	
--------	-------------------	-----	-----	---	---	---------	--

Test Step (PLC Cycle: 22)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.SequenceStates		Int		0	Success	

Test case "Outfeed line empty after bad product and push off band"

- **Author:** Flamur Breznica
- **Execution date:** 2019-12-12T15:36:59.6529347Z
- **Result:** Success

Test Step (PLC Cycle: 1)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.PLCLnfeed		Bool		true	Success	

Test Step (PLC Cycle: 5)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedOpen		Bool		true	Success	
INPUT	Simulation_DB.actualUnderPressure		Real		-1.0	Success	
OUTPUT	db.lineEmpty	Bool	Bool	False	false	Success	

Test Step (PLC Cycle: 10)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedFilling		Bool		true	Success	
INPUT	Simulation_DB.flowmeterSensor		Real		100.0	Success	
INPUT	Simulation_DB.sealerNewTemperature		Real		100.0	Success	
OUTPUT	fb_main_DB.Product.fillLevel	Real	Real	100	100.0	Success	
OUTPUT	fb_main_DB.Product.underPressure	Real	Real	-1	-1.0	Success	

Test Step (PLC Cycle: 15)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.reachedSealingControl		Bool		true	Success	
OUTPUT	fb_main_DB.Product.sealingTemp	Real	Real	100	100.0	Success	
OUTPUT	fb_main_DB.Product.isOpen	Bool	Bool	False	false	Success	

Test Step (PLC Cycle: 20)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	fb_main_DB.Product.badProduct		Bool		true	Success	
INPUT	db.reachedOutfeed		Bool		true	Success	
INPUT	db.simulate		Bool		true	Success	
OUTPUT	db.pushOffBand	Bool	Bool	True	true	Success	
OUTPUT	fb_main_DB.Product.badProduct	Bool	Bool	True	true	Success	

Test Step (PLC Cycle: 21)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	Simulation_DB.movingPush		Real		100	Success	
OUTPUT	db.lineEmpty	Bool	Bool	True	true	Success	
OUTPUT	db.SequenceStates	Int	Int	0	0	Success	

Test Step (PLC Cycle: 22)

- **Result:** Success

Direction	Name	Type	Expected type	Value	Expected Value	Result	Message
INPUT	db.SequenceStates		Int		0	Success	