



Automatic GUI testing on embedded display module

Tobias Österberg

Bachelor's thesis
Electrical Engineering
Vaasa 2014



BACHELOR'S THESIS

Author: Tobias Österberg
Degree Program: Electrical Engineering
Specialization: Automation
Supervisor: Matts Nickull

Title: Automatic GUI testing on embedded display module

Date: 10.11.2014 Number of pages: 42 Appendices: 1

Summary

This thesis is written for Platform Software, A&C, R&D, 4-stroke, Ship Power, Wärtsilä Finland Oy during the summer of 2014.

This thesis is about automatic GUI testing and how this can be implemented on embedded systems. The thesis consists of an investigation in the possibility to implement automatic GUI testing for Wärtsilä's LDU. The LDU is a display module for Wärtsilä's engine control and monitoring system UNIC.

The goal is to investigate the possibility to automate the GUI testing on the LDU. This is done by comparing different tools, testing environments and techniques to do the testing on the LDU. Thereafter a proof of concept is made so that the possible future development of the GUI testing has something to start from.

vTask Studio was chosen as a suitable tool to include in the proof of concept. Based on the proof of concept vTask fulfills our expectations and is a worthy candidate for the future development of the automatic GUI testing.

Language: English Key words: automatic testing, GUI

EXAMENSARBETE

Författare: Tobias Österberg
Utbildningsprogram och ort: Elektroteknik, Vasa
Profilering: Automation
Handledare: Matts Nickull

Titel: Automatiserad testning för grafiskt användargränssnitt på inbyggd bildskärmsmodul

Datum: 10.11.2014

Sidantal: 42

Bilagor: 1

Abstrakt

Examensarbetet är gjort åt Platform Software, A&C, R&D, 4-stroke, Ship Power, Wärtsilä Finland Oy under sommaren 2014.

Detta examensarbete handlar om automatiserad testning för grafiska användargränssnitt och hur detta kan tillämpas på inbyggda system. Examensarbetet innehåller en undersökning kring möjligheten att implementera automatiserad testning för Wärtsiläs LDU. LDU:n är en bildskärmsmodul för Wärtsiläs motorstyrnings- och övervakningssystem UNIC.

Målet är att undersöka möjligheten för automatiserad testning av LDU:ns grafiska användargränssnitt. Detta görs genom att jämföra olika verktyg, testningsmiljöer och tekniker på hur testningen kan göras på LDU:n. Vidare ska en konceptvalidering utföras så den möjliga framtida utvecklingen av testningen har något att utgå ifrån.

vTask Studio valdes ut som ett lämpligt verktyg att ingå i konceptvalideringen. Genom konceptvalideringen framkom det att vTask lever upp till förväntningarna och är en värdig kandidat för fortsatt utveckling av automatiserad testning.

Språk: Engelska

Nyckelord: automatiserad testning, grafiskt användargränssnitt

Contents

1	Introduction.....	1
1.1	Wärtsilä.....	1
1.2	Purpose and goals.....	2
2	Theory of automatic GUI testing.....	2
2.1	Graphical User Interface.....	3
2.2	Testing.....	3
2.2.1	Unit testing.....	4
2.2.2	Functional testing.....	4
2.2.3	GUI testing.....	5
2.3	Benefits and risks of automatic GUI testing.....	5
2.3.1	The benefits of automatic testing.....	6
2.3.2	The risks of automatic testing.....	7
2.3.3	Limitations.....	8
2.4	Unsuitable situations for automatic testing.....	9
2.5	Test planning for GUI testing.....	10
2.5.1	Test specification and test design documentation.....	10
2.5.2	Advantages of a thoroughly made test documentation.....	11
2.6	Characteristics of well-designed automatic GUI testing.....	12
3	Selecting a suitable automatic GUI testing tool for the LDU.....	13
3.1	The LDU - The Local Display Unit.....	14
3.2	Testing technique.....	15
3.2.1	Image based testing.....	15
3.2.2	Object based testing.....	18
3.2.3	Selecting suitable technique.....	19
3.3	Testing environment.....	20

3.4	Testing tools.....	21
4	Testing with vTask Studio	24
4.1	Basic usage	24
4.2	Compatibility with Jenkins	26
4.3	Designing of test cases.....	27
4.3.1	Home page layout test	28
4.3.2	Shortcut functionality test.....	28
4.4	Explanation of scripts.....	29
4.4.1	Test initiation script	29
4.4.2	Home page layout script.....	30
4.4.3	Shortcut functionality script.....	31
4.4.4	Test result TAP-files	36
5	Results.....	37
6	Discussion	40
7	List of Sources.....	41
	Appendix 1	

Abbreviations

R&D – Research and Development

A&C – Automation and Controls

GUI – Graphical User Interface

LDU – Local Display Unit

UNIC – Wärtsilä Unified Controls

API – Application Programming Interface

IDE – Integrated Development Environment

KVM switch – Keyboard, Video and Mouse switch

VNC – Virtual Network Computing

TAP – Test Anything Protocol

XML - Extensible Markup Language

1 Introduction

This thesis was commissioned by Wärtsilä Corporation. My supervisor was Leif Strandberg who is manager of the Platform Software group. The Platform Software group is a part of the A&C group, which is a part of the R&D group for the 4-stroke group of Ship Power, which is one of the three major units in Wärtsilä. The background of the thesis started when I in my first year as trainee at Wärtsilä manually tested the LDU. The following year as trainee I came in contact with automatic GUI testing with Squish for one of Wärtsilä's software. At that time I got interested in automatic GUI testing and got the idea of writing a thesis about automatic GUI testing on the LDU. Automatic GUI testing on the LDU has been considered but no resources have been put into it.

The main issue is that more testing needs to be done to increase confidence in the system. The Testing of the LDU's GUI has always been manual labor and to automate some of the more basic repetitive tasks would give the testers more time to spend on more advanced tasks.

1.1 Wärtsilä

Wärtsilä is a global company with 18 700 employees that operates in more than 200 locations in 70 countries with net sales of EUR 4.7 billion. Wärtsilä produces complete lifecycle power solutions for the marine and energy markets. Wärtsilä is divided into three major units, which are Ship Power, Power Plants and Services /1/



Figure 1. Wärtsilä logo. /2/

The Ship Power unit provides solutions for machinery, propulsion and maneuvering for the marine industry. Wärtsilä supplies engines and generating sets, reduction gears, propulsion equipment, control systems and sealing solutions for all types of vessels and offshore applications /1/

The Power Plant unit provides solutions for the decentralized power generation market. They provide power plants for base load, peaking and industrial self-generation purposes as well as for the oil and gas industry. /1/

The Service unit provides customer support through the lifecycle of the installations. They give solutions for service, maintenance and reconditioning for both ship machinery and power plants. Conditioning and maintenance for other engine brands are also provided. /1/

1.2 Purpose and goals

The purpose of this thesis is to do more regression testing with the help of automatic GUI testing. This will increase confidence in the system by ensuring changes to the code, such as new features or bug fixing, don't make another part of the code corrupt. The goal of this thesis is to make a proof of concept and evaluate a method to do automatic GUI testing on the LDU based on theory about GUI testing. In the future the result from this thesis will hopefully be a starting point for the development of the automatic GUI testing for the LDU.

2 Theory of automatic GUI testing

In our daily life we are surrounded by computers and software. We interact with different programs in the industry, educational institutions and other enterprises and organizations. We are dependent on software for product development, production, marketing, support, services and management. This puts great pressure on the software development because it needs to be cost efficient and with high quality. To ensure high quality, testing of the software in as many ways as possible is required. Before a program can be released it has to go through different test cases during and after the development

to ensure it is working as intended. More testing can be executed by letting the computer perform testing on the developed software by using automatic testing tools. These tools are usually commercial programs. They generally offer a complete suite for testing and managing testing or specifically developed testing frameworks for the tested software. /3/

2.1 Graphical User Interface

The first programs made, where the user was able to interact with the program, were command-line driven. The user had to remember what commands to give and when to give them. More advanced applications could give the user available commands to enter and prompt the user for next command. To increase usability, GUIs were introduced. /3/

The Graphical User Interface, or GUI for short, is what the user finds on the screen when the program is started. The GUI consists of different components or objects like windows, menus, icons, pointers, text and images. By using a mouse and keyboard or touchscreen the user is able to control the software according to the user's will by clicking buttons and entering various keystrokes. Since usability is a very important factor for a good program, lots of time is put into designing and programming the GUIs. This means that a lot of testing is required. Additionally to usability testing, it is also needed to ensure for example that each button executes the right functionality, as well as each object that should be visible in the GUI is in fact visible. /3/

2.2 Testing

There are many kinds of testing and many ways to execute tests. There are also different priorities for the different kinds of testing. All this is usually entirely up to the software that is developed. Figure 2 shows an overview of how testing should be structured for the software of the LDU. This is easily displayed as a pyramid to briefly show how to prioritize and how the three different testing types tie together. All testing should have a steady foundation of unit testing, which should be up to 80 % of the total testing done. As seen in the pyramid GUI testing should only take up to 5 % of the total testing. In practice the different testing types are not always clearly separable. Depending on the method of

testing some types of testing are combined. This applies especially to GUI and functional testing, which can usually be and sometimes preferably be executed at the same time.

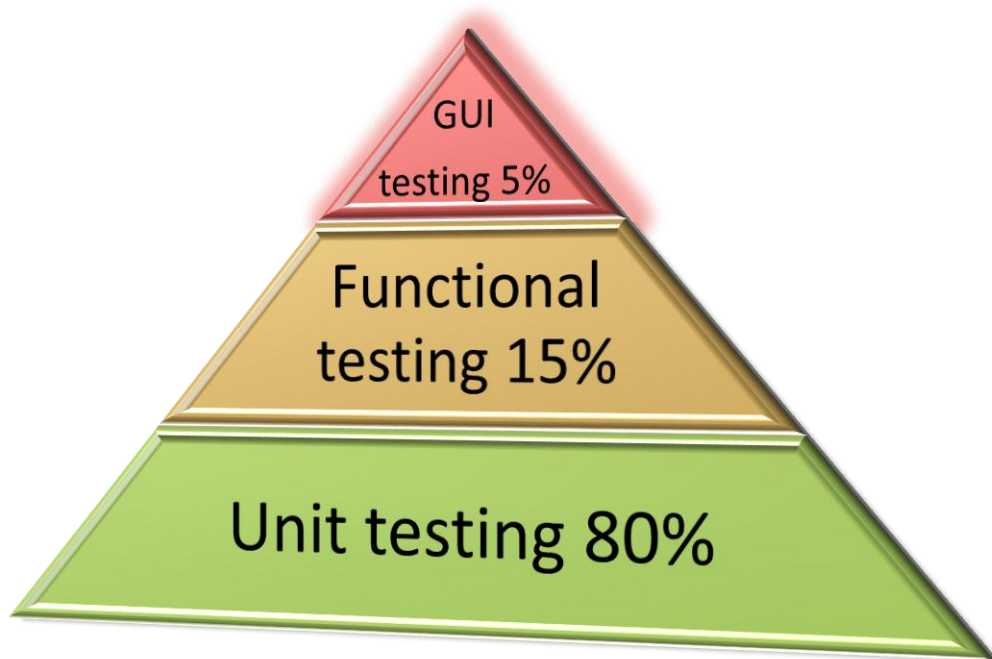


Figure 2. Pyramid showing how to prioritize and organize testing.

2.2.1 Unit testing

Unit testing is testing done on a single piece of code like a class or function. The purpose of the testing is to check some assumptions about the behavior of the code under test. The testing is executed by an automated piece of code, which is usually written using a unit-testing framework. Unit testing can be done after or before the software is written. Writing the test before the software is written is called test-driven development.

Making a unit test for a function that performs an XOR on two input values and returns the result is quite simple. A simple validation of the function would be to enter the four different input combinations to the function and compare the result with the expected result. /4/

2.2.2 Functional testing

Functional testing is verifying that the program or module behaves according to the requirements or specification by testing specific features or smaller parts of the program.

Functional testing only observes the output of a specific input and does not analyze the code, what functions are used or the other internals of the program. Functional testing is sometimes referred to as black-box testing or behavioral testing.

An example of a functional test is to test an exit warning feature in a text editor. The specific input of the test is clicking the close button after a text has been changed. This should lead to the desired output, which is a warning message to inform the user that there are unsaved changes. /5/

2.2.3 GUI testing

GUI testing is testing done to verify that the objects of the GUI are correctly displayed and when combined with functional testing also to verify that the objects activates the right features. GUI testing verifies the visibility, location, alignment, font, color and look of the objects. An example of a GUI test is to verify from the example of functional testing that the warning message that appears is on top of the program, contain the correct information and have the correct buttons. /3/

2.3 Benefits and risks of automatic GUI testing

The main reason behind the necessity for automatic software testing and automatic GUI testing is *“the need for speed”* as Linda G. Hayes explains in her book *The Automated Testing Handbook*.

The need for speed is practically the mantra of the information age. Because technology is now being used as a competitive weapon on the front lines of customer interaction, delivery schedules are subject to market pressures. Late products can lose revenue, customers, and market share. But economic pressures also demand resource and cost reductions as well, leading many companies to adopt automation to reduce time to market as well as cut testing budgets. /6/

Releasing defective software to the market may have tragic consequences. Depending on the software released the company could be responsible for customers’ lost production when the delivered software failed. Another case could be a defective engine control system on a ship in the middle of the ocean, which may require an expert to fly to the

ship to repair the system. The compensation to the customer can cost millions and the reputation of the company is damaged, which is usually the most serious issue. The main reason automatic testing exists is to ensure more testing is done, not to save money spent on testing. Furthermore, if there aren't enough resources spent on testing to begin with, adding automation will not help to increase confidence in the system noticeably. /6/

2.3.1 The benefits of automatic testing

There are many reasons especially for larger software companies to spend time and money on automated software testing and automated GUI testing.

- ❖ **Run existing or regression tests on a newer version of the tested program.** Running regression tests is important because programs change during the course of their lifetime and new features are added. A 10 % change in the code still need 100 % of the features tested to ensure that the new code doesn't affect the old features. Unfortunately, test engineers often have to prioritize testing the new features rather than the old features when time is short. This is why it is a clever move to make the automatic testing do regression tests. The test cases should not take a long time to adjust for the updated program when they have already been executed on an earlier version of the program. /7/ /6/
- ❖ **Run more tests more often.** Because of the efficiency of automatic testing, more test cases can be run in a shorter time than corresponding manual testing. As a result, this will increase the confidence in the system. /7/
- ❖ **Run test cases that are impossible or hard to replicate manually.** There are tests for the GUI which are hard or impossible to verify manually. For example if a test of a button is wanted on a GUI, which triggers an event not visible on the GUI, it's hard for the test engineer to verify the test. Depending on testing method, an automatic testing tool doesn't have that limitation and can check if the event has been triggered. /7/
- ❖ **Improved usage of resources.** The repetitive and tedious test cases such as entering the same input over and over are easily implemented as automated

testing. It will ease the workload and increase morale of the staff when they can use the time to do more advanced testing or designing more test cases. /7/

- ❖ **Consistency and repeatability of tests.** Automatic tests will always be executed in exactly the same way every time. This will increase consistency to levels unthinkable by manual labor. /7/
- ❖ **Reuse of tests.** The time put on creating automatic tests can be distributed over the many executions of that test compared to the manual tests. Manual tests are usually reused much less. Tests that are reused all the time are worth spending time on. /7/
- ❖ **Earlier release.** Once the automatic testing is running at full capacity the testing time for a version of the program is greatly reduced. Ultimately it will make the program ready for release faster. /7/ /6/
- ❖ **Increased confidence in the software.** When the personnel are aware of the great deal of tests that the program has gone through successfully, it will increase confidence in the program. As well as that there won't be any horrible surprises after the software is released. /7/

2.3.2 The risks of automatic testing

As there are benefits there are also risks for the company to pay attention to before and after adding automatic testing.

- ❖ **Unrealistic expectations.** The industry has a tendency to incorporate any fresh and new solution and think it will solve all problems. Unfortunately this also applies to automatic testing. The human being usually wants to believe that everything is fine when a new tool or solution is put into use. Automatic testing will not be the solution for all testing. /7/
- ❖ **Expecting automatic testing to compensate for poor testing practice.** If the testing practice is poor there is no reason to add automatic testing. Poor testing practice means that tests are unorganized, documentation is inconsistent and tests are not good at finding bugs. The focus should then be on improving the effectiveness of testing, rather than improving the efficiency of poor testing because; *“Automating chaos just gives faster chaos”*. /7/

- ❖ **Expectation to find lots of new bugs.** A new test is probably going to find a bug or two the first time it's executed. The following executions will probably not find any new bugs unless the software is updated. The focus of automated testing is to do regression testing. Regression testing means proving that the software still works as intended even if something has been updated in a different part of the software. /7/
- ❖ **False sense of security.** Automatic testing will increase confidence in the system. However, just because the tests are successful doesn't mean that the software is without defects. The test cases may be incomplete or there might be problems with the design of the test cases. Always keep an eye open for problems because no software is perfect. /7/
- ❖ **Updating of automated tests takes too much effort.** Usually when the GUI is updated there are probably some of the automated tests that need updating. The effort to update the automated tests should be low. When the effort of continuously updating the automated tests takes longer than doing them manually, it will usually be the end of the automated test initiative. /7/
- ❖ **Technical problems.** As with all software there is no such thing as complete immunity against bugs and automatic testing tools are no exception. The tool might not work properly with other applications and software used. The commercial tools are unfortunately usually large complex products, which require great technical knowledge. In addition to all this, there is still the tested program which might not be designed or built with testability in mind. /7/

2.3.3 Limitations

As the previous chapters explain, automatic testing could be a viable solution to some of the testing difficulties and that is not without risks. Additionally, automatic testing has noticeable limitations.

The main limitation is that it can't replace manual testing. It is impossible to automate everything for technical or economical reasons. It is usually not feasible to automate tests that are run very rarely or where the GUI changes drastically with every version of the program. In the same way it is not possible to automate tests that concern for example

usability or esthetic appeal. Another thing to have in mind is that manual testing is the main source of finding new bugs, not automatic testing, which just re-runs previous tests and will probably not find new bugs. /7/ There is one exception and that is if the test tool does so called monkey testing, which is best described as randomly generated keyboard inputs and random mouse clicking all over the GUI. /3/

The added complexity of automatic testing will also limit the effectiveness of testing. However, in the long run automatic testing will pay off regarding efficiency only if it's developed correctly so it's not crippled every time the tested program is updated. If it requires a high effort to make new tests or changing the old tests, it may restrict the development of the tested program. Especially when considering making minor changes to the tested program. /7/

Finally, tools have no imagination. Because automation tools are just software that does exactly as instructed and nothing else, some important things can be missed. Manual testers can see that something is wrong with a feature that's not supposed to be tested in that case. The manual tester can then edit the test case to also test that feature next time the test case is executed. Another limitation because of lack of imagination is also the issue of handling unexpected events. For example network connection issues will most likely fail a test controlled remotely, while the human tester can easily overcome such problems by using the computer where the tests are performed. /7/

2.4 Unsuitable situations for automatic testing

Usually if one has taken into consideration the benefits, risks and limitation with automatic testing it would be acceptable to try it out. However, there are some situations when it is not worth the investment.

Certain programs are unstable by design. This isn't referring to a program's tendency to crash. It is rather referring to the program's high configurability or difficulty to control the inputs of the program. This makes it difficult to have an appropriate estimate of output. If this is the case it will be impossible to automate because of the varying output. Usually the testing tool wants full control of the program. This means that an appropriate testing environment must be created where the tool has full access to all required data. /6/

The main reason not to automate is if the testers are inexperienced, temporary or have insufficient resources to do what it takes to develop automatic testing. If the testers are inexperienced with the tested program and the testing tool, the value of the results of the automatic tests is doubtful. The automatic tests are only as good as the person who created them. Automatic testing should preferably be done by experts to ensure efficiency. On the other hand, experts are more expensive, which makes it tempting to hire them temporarily as consultants. However, automatic testing should not be performed when using temporary testers. The initial investment in learning the tools and test cases is quite high and therefore automatic testers should be involved in the long term and not temporarily. It's more efficient to let the experts provide support to develop test cases and let the permanent staff handle the automation. If there are no resources for manual testing or a permanent staff dedicated to testing there is absolutely no reason to automate any testing. /6/

2.5 Test planning for GUI testing

The foundation for a successful manual and automatic testing effort is the same as the foundation of a successful software development project, which is sufficient planning. It can be tempting to do brief planning for automatic GUI testing, because testing is supposed to be easy and the test code will not be available to the customer. This is why it seems like a clever idea to save time by reducing the time spent on planning. However, in reality test planning and design are hard to do well and usually the test tool is more complicated than it may look. Writing and debugging test code may be difficult. Therefore it's not recommended to do the planning as the automation project goes along. Otherwise the automation solution imagined in the beginning of the project may turn out to become something completely different because of the lack of planning. /7/ /3/

2.5.1 Test specification and test design documentation

Usually the documentation for testing is split up into two major parts, which are test specification and test design. This thesis will not include these documents since this thesis is intended as a proof of concept and not a final product. /7/

The test specification for testing is similar to the functional specification for the application. This document addresses the high level aspects like organization, testing scopes and schedules. The template for this document is usually designed by the companies according to their policies. A good template is given in the IEEE Standard 829. /7/

The test design is a blueprint for the developers. This documentation describes exactly what will be tested and how it will be executed. The basic version of a test design should at least include a: /7/

- ❖ **Test name or ID.** To allow easy identifying of what category of test it is, a unique identifier is needed.
- ❖ **Test purpose,** which describes briefly what the test is supposed to achieve
- ❖ **Test method,** which is a description of what steps to take to perform the test. The description should be clear enough so that it is possible to perform the test manually with the provided information.
- ❖ **Pass/fail criteria,** which describes the expected result of a successful or failed test

2.5.2 Advantages of a thoroughly made test documentation

Making the test documentation thoroughly according to the previously mentioned system has several advantages.

- ❖ **Easier review of the test plan.** There is usually not enough time to do all tests and compromises have to be made regarding quantity and complexity of the tests. Good documentation is therefore needed for intelligent review of the test plan where the test team can easily make the compromises. /7/ If the test design is poorly written it's hard to understand what the test cases are expected to do. This ironically tends to make it more difficult to remove a test case because it could be important. /6/

The easily reviewed documentation also helps speed up the process of deciding what test cases to automate. From the test designs it's easy to see how many test cases that are suitable for automatic testing. /7/

- ❖ **Easier test maintenance.** Good test documentation gives a good overview of the test cases, which should be included in the test code to increase maintainability. High maintainability saves time, which makes testing more cost efficient. /7/
- ❖ **Bug finding.** The thoroughly made test design process usually finds more bugs than the actual testing. This is because the test cases are run for the first time. /7/

2.6 Characteristics of well-designed automatic GUI testing

After the review of the test documentation and test cases to automate have been chosen it's time to start building the automatic tests. There are some basic attributes for successful and effective execution of automatic test cases. These are maintainability, modularity, robustness, optimization and thoroughly done documentation.

Test cases should be maintainable. To be able to accumulate tests, automatic test cases should be easy to maintain since on average 25 % of an application is modified each year. If the required time to adjust the test scripts accordingly isn't reasonable, the test case will become obsolete. This means that rather than increasing test coverage over time it will decrease, especially if new features are introduced. To increase maintainability the test framework or testing tool should be easy to adjust when source code changes and the testing team should be aware of upcoming source code changes and new features. /7/ /6/

The test suite should be optimized. More test cases are not always better. More test cases require more time to develop, maintain and execute. Investigating the test cases is recommended for achieving balance of confidence in the system and the number of tests to maintain. /6/

Test cases should be independent. All test cases should on their own go from a base state to the state where they can perform their task and then go back to the base state without being dependent of another test case's result. Otherwise it will be harder to track down the actual reason that makes the tests fail. Therefore, when possible, the test cases should be able to be executed independently without the need to perform them in a specific order. /7/ For example, if one test case tests the deletion of data it should not be dependent on another test case that tests the creation of data. The test case should

instead delete data that already exist in the base state or create own data and then do the test. /6/

Test cases should be robust. If there is something failing in the application that's not a part of the test, such as a crash, error message or entering a state that was not expected, the test case should log that the test failed and abort the rest of the test and try its best to reset the application to the base state so the next test can be executed. /7/

Test cases should be well documented. Documentation is explained in chapter 2.5.

3 Selecting a suitable automatic GUI testing tool for the LDU

There are a couple of things to take into consideration when choosing the most suitable way to implement automatic GUI testing for the LDU. In automatic GUI testing there are two main techniques available that are interesting in this case. These are image based testing and object based testing. There are two possibilities regarding the test environment, testing on standalone version of the GUI in Windows or testing on hardware module. Depending on chosen environment and technique there are different tools available to use.

3.1 The LDU - The Local Display Unit

The Local Display Unit or LDU for short is an embedded display module in the UNIC system running embedded Linux and the GUI uses QML as programming platform. UNIC is an engine control and monitoring system developed and used by Wärtsilä. The UNIC system consists of different types of hardware modules that control and monitor the engine and PC software to update software in the modules and monitor the engine.

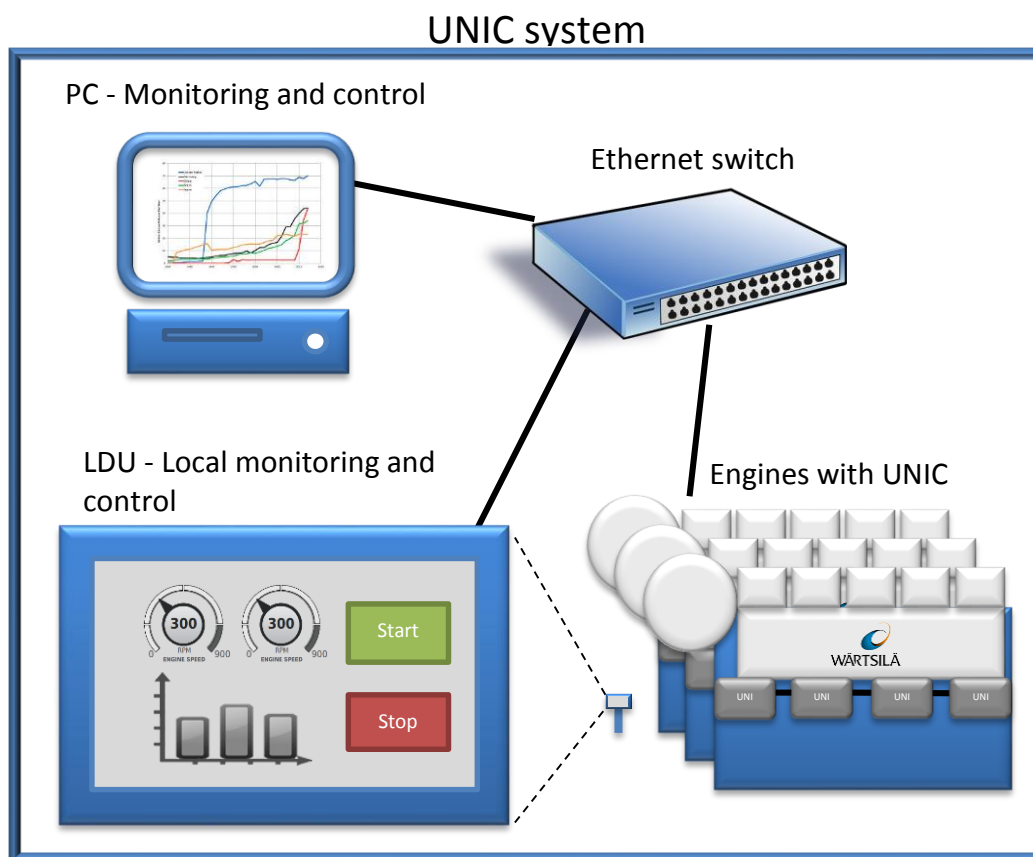


Figure 3. An example of a UNIC system.

The purpose of the LDU is to offer a local user interface for the engine limited by user rights. It's usually placed close to the engine where the user is able to start or stop engines and increase or decrease engine speed. Other actions include checking software versions and configuring Ethernet settings. The LDU also offers several options for monitoring the engine. For monitoring numerical values such as temperatures and pressures there are alternatives such as bars, gauges, trend charts and plain numbers. Monitoring events and actions such as engine start, communication failures or values

outside their limits can be done in the log. The LDU is highly configurable and therefore requires a great deal of testing. /8/

3.2 Testing technique

The majority of modern GUI applications are built on a programming platform, such as Java or Qt, to allow developers to build complex GUIs with the help of libraries. The platform paints the GUI according to the application and sends it to the OS screen image buffer, which then sends the whole frame to the display. To control the GUI the application registers the keyboard and pointer events delivered by the OS from the mouse and keyboard. /9/

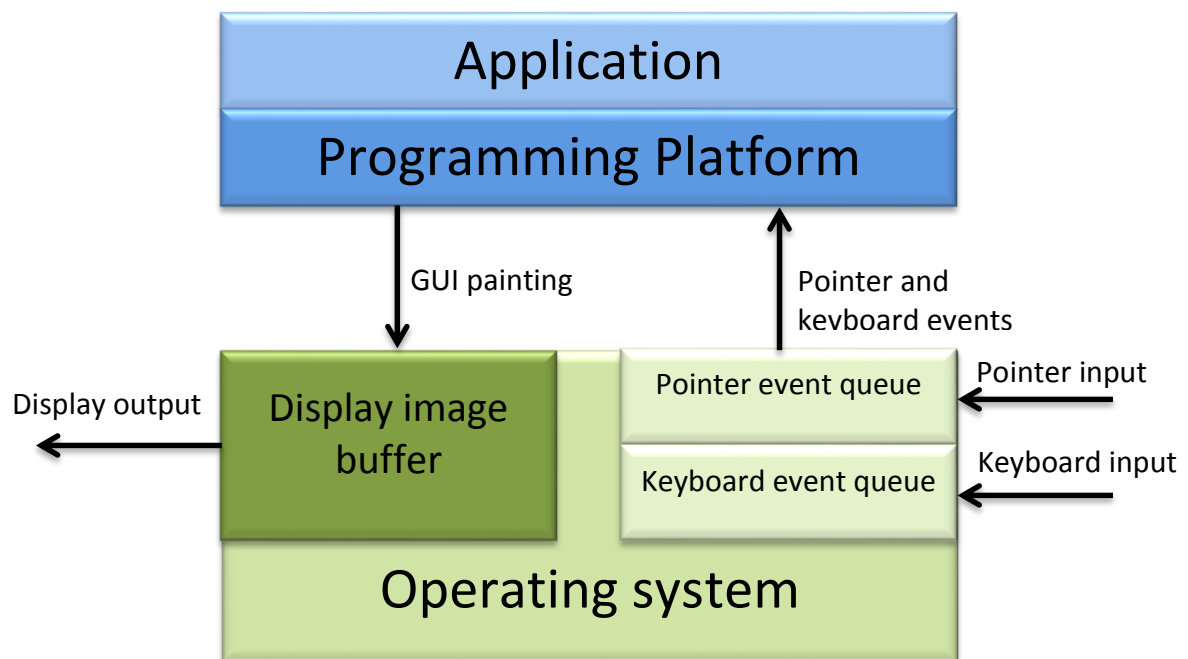


Figure 4. Architecture of an application focused on the GUI aspect.

Automatic testing tools can roughly be classified into two techniques concerning the method to approach the GUI of the tested program. Both methods have their advantages and disadvantages depending on the software developed.

3.2.1 Image based testing

Image based testing automates on OS level or through another layer to allow access to the tested program with virtualization or remote desktop software. The image based tool

makes keyboard and pointer input events to control the tested program, which is in the same way as a user would control the tested program. The GUI of the tested program is accessed with screenshots through the display image buffer. Verification is usually done with the screenshot through image comparison, object recognition or text recognition.

/9/

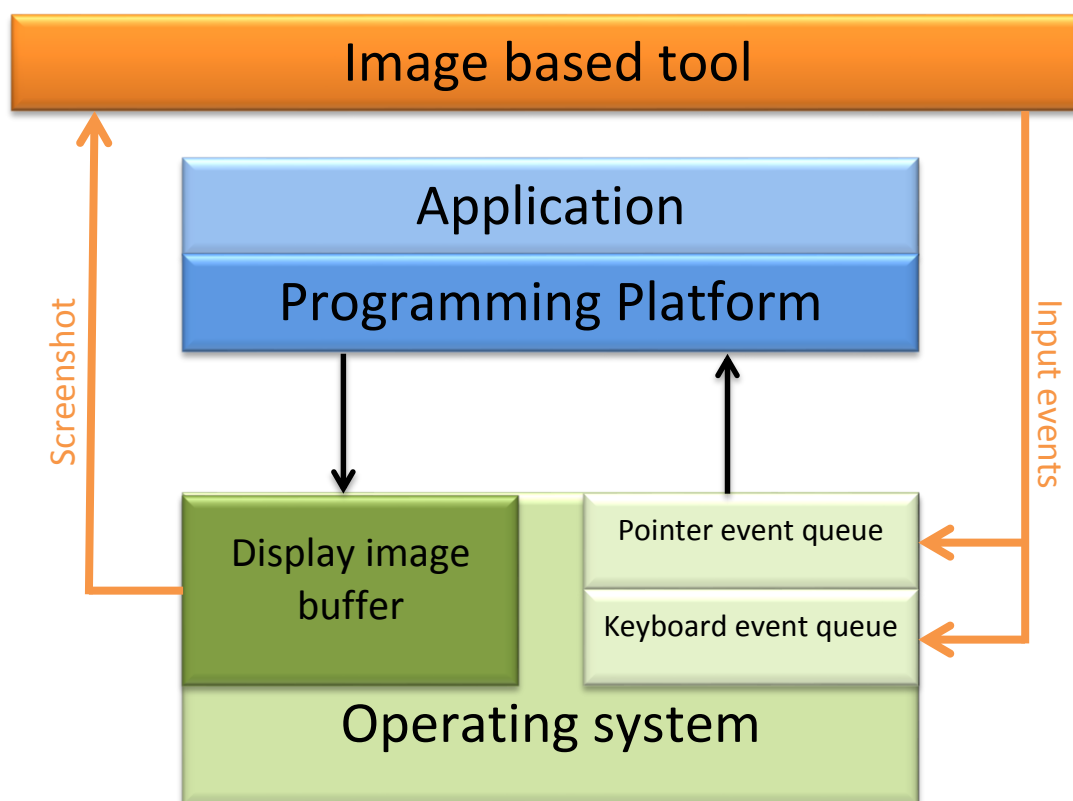


Figure 5. Image based testing approach.

The main advantages and disadvantages of image based testing are /9/:

- + **End user approach.** The tool will see what the end user will see and controls the tested program the same way as an end user. This method will be the most accurate way of testing real life use of the application.
- + **Application technology independence.** The image based tool will test the tested program, no matter which programming platform or platforms that are used in the program. As all inputs and outputs are handled on OS level, the programming platform becomes irrelevant for GUI testing.

- + **Clean test environment.** Image based tools use only the features included in the OS and do not need any third party libraries or programs.
- + **Simple automation.** Usually the image based tools are easy to learn and require no experience of the application's programming platform.
- **No access to object properties and data or functions.** Image based testing will not have access to reading the objects or functions connected to the object. This is not necessary a limitation if the objects' functionality can be verified through the GUI.
- **Sensitive to test environment changes.** Image based testing usually requires a stable test environment to function optimally. If there is a change in resolution or colors or there are windows popping up there can be failed tests.
- **Migration risks.** There may be problems if the test environment changes, for example from a PC to a mobile phone, which may require changes in all test scripts especially images that are compared to screenshots.
- **Sensitivity to graphical effects.** Because the image based tool is based on image recognition it's sensitive to graphical effects such as anti-aliasing.

3.2.2 Object based testing

Object based testing relies on a tight integration with the programming platform. These kinds of tools have access to libraries of the programming platform and can identify the objects and read their properties and in that way check that the GUI is correctly configured. Interaction of the GUI is done either through the platforms APIs, with simulated clicks and other interactions, or through pointer and keyboard events on OS level. Verification is done by checking the existence, state or properties of the tested object. Object based testing makes it possible to do more inclusive functional testing into the GUI testing. /9/

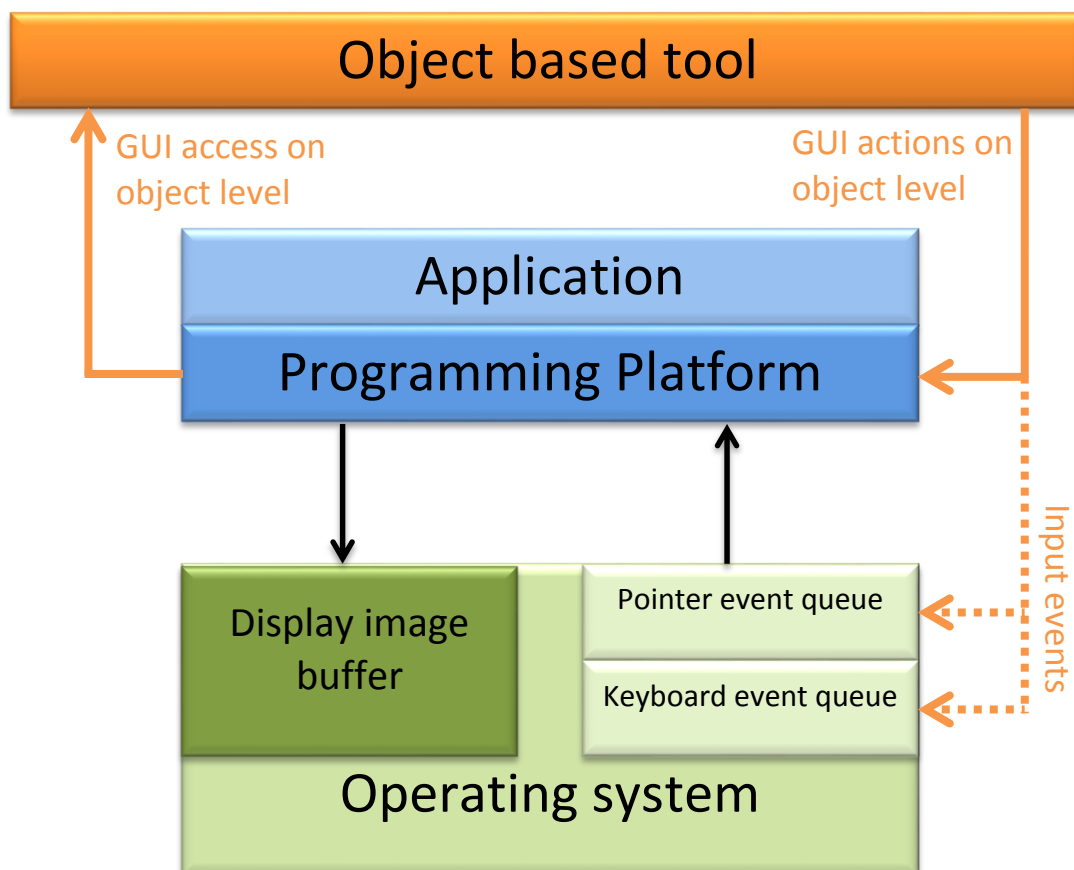


Figure 6. Object based testing approach.

The main advantages and disadvantages for an object based approach are /9/:

- + **Platform integration.** Thanks to the tight integration with the programming platform the tool is able to exploit features in it for performing a deeper level of testing that image based testing is not able to do.

- + **Robustness.** Test scripts are very robust thanks to the ability to identify properties and perform actions of individual GUI objects.
- + **Good verification.** Verification is more precise and robust than image based testing thanks to the ability to read the properties of the object directly.
- + **Easy to understand.** If the test engineer is familiar with the programming platform it's easy to understand and perform efficient testing.
- **Bound to a specific programming platform.** If there are GUI objects in the tested program using a programming platform the object based tool is not supporting, it will not be able to test these objects. This could happen when using third party applications in the GUI. Nevertheless, most of the more expensive tools usually support a lot of programming platforms to avoid this issue.
- **Not able to test all features.** Usually the object based tools are not able to test all features that are supported by the programming platform or if there are custom made GUI features.
- **Inability to detect GUI layout errors.** The object based tool can't detect problems with the layout, for example objects overlapping each other. It can only check if it exists but not that it is actually visible on the GUI.
- **Upgrade risks.** Updates to the programming platform might give compatibility issues until the tool is updated to support the new version, which may delay testing and may further delay release dates of the application.
- **Higher qualification requirements on test engineers.** The added complexity of object based testing requires higher experience in both testing and programming, which may or may not increase automation cost. Object based testing have a higher learning curve than image based.

3.2.3 Selecting suitable technique

Selecting the most suitable technique for GUI testing of the developed application is the key for successful and efficient testing. When selecting the technique for automatic GUI testing there are four main factors to consider before making a decision. /9/

- ❖ **Required test depth.** Is there a requirement for the testing tool to be able to do some functional testing? In other words, does the tool need to be able to access GUI components? Is a more end user focused testing preferred?
- ❖ **Technology scope.** Are there or will there be two or more different technologies included in the developed application? Are all technologies supported by the tool?
- ❖ **Migration and upgrade plans.** Will the test environment be stable such as a dedicated PC or server or are there requirements for the tool to be able to migrate to another test environment or programming platform?
- ❖ **QA resources at hand.** Are the testing engineers experienced with the application and the programming platform of the application? Is there time to train the resources at hand to master the programming platform?

The next step to take is to take into consideration these general factors. Based on these factors a tool will be chosen and compared to the situation with the LDU. The main reason to try image based testing is because of the desired end user approach. Functional testing is not usually required because of the fact that the majority of the functionality of the LDU is handled on another module in the UNIC system. This module is thoroughly tested with automatic functional testing. The simplicity of the image based approach is appealing because of the possibility to put anyone on the test scripting without the need to educate the testers in the programming platform. Factors speaking for an object based approach are possible test environment change if there is time and requirements for it but otherwise the testing will be done on a dedicated PC or server. The use of only one programming platform also supports the use of an object based approach. According to these factors and discussions with people responsible for the LDU development, the most suitable technique to try out was decided to be the image based approach.

3.3 Testing environment

When it comes to testing environments there are two possible solutions. The options are to execute testing on a standalone version of the GUI on PC or testing on LDU hardware.

Testing on hardware module would be the preferred environment as it would bring the testing tool as close as possible to the end user's way of controlling the GUI. Running the

tests on real hardware will notice possible bugs in the GUI caused by display drivers and notice if there are long processing times for an action. Unfortunately, testing in this environment is difficult because of the limited connectivity of the LDU, which is limited to a USB and Ethernet port, which excludes a KVM switch solution. Running a VNC server on the LDU may be the improper solution because of the added software, which is undesirable in testing of embedded systems. Executing testing on the LDU hardware may make it unavailable or not easily accessible for other testing such as hardware or manual testing.

The other test environment alternative is a standalone version of the GUI on a dedicated PC or server. This will ensure a very stable environment and decrease complexity of testing as a desktop application is much easier to control than an embedded application. Because the programming platform is QML, the application will look identical and perform identically in both the desktop and the embedded environment, which makes the desktop version of the GUI an acceptable solution. As no LDU hardware is needed, the application is always accessible. The main drawback with testing on a PC is that the application is driven by more powerful hardware, which makes it hard or impossible to test responsiveness. This will require some manual testing on the LDU, which is not necessarily undesired as the responsiveness and look of the LDU is best done manually.

By taking these facts and discussions with people responsible for the LDU development into consideration, it was decided that the standalone version of the GUI on a PC was the preferred solution. The ability to do testing without access to the LDU hardware and having the comfort of executing the tests on a PC were considered desirable.

3.4 Testing tools

Since the testing technique and testing environment have been decided, the next step to take is to compare and choose a testing tool according to requirements. The basic requirement for the tool is compatibility with Jenkins, which means there must be a way for Jenkins to manage testing such as starting the test execution by command line and a way for the tool to send test reports to Jenkins. Jenkins is an application for monitoring executions of repeated jobs, such as building software and automatic tests. This is

described in more detail in chapter 4.2. Features not required but beneficial are VNC support, text recognition, ease of use and pricing. Short facts of the testing tools taken into consideration in this thesis can be found in appendix 1.

The higher the price the more features and other benefits are included in the tool. The cheaper tools lack more advanced features such as text recognition, which is a good feature to have if the graphics is in the development stage. Using VNC is not desired yet because of the need to install software on the LDU. However, this might change later, which makes VNC support an extra feature that is nice to have. The main advantage of the higher cost solutions is the support that may be lacking in the cheaper tools. Price may matter, but the main factor is easy and infrequent need of maintenance. If maintenance requires little or no effort when using a more expensive tool it will probably be cheaper in the long run than a cheaper tool that needs more effort in maintenance.

After discussion with people responsible for the LDU about the facts concerning the tools mentioned in appendix 1, we decided to try out vTask Studio. This tool caught attention because of its lack of scripting language. Instead of scripting using a programming language it is done by drag-dropping pre-made function blocks that are highly configurable and easy to learn. Another distinctive feature is the ability to compile every test script into a compact executable file, which will enable testing without the need of libraries and a license of the tool installed on a dedicated PC. With a price of only 40 € the tool is essentially free. However, there may be problems to get efficient support as the support is handled by user forums.

Compared to the other tools vTask is easier to use than the freeware tool Sikuli that would need some complementing libraries to get the same functionality as vTask. Sikuli's main feature is the ability to adjust the tool to the project's desire as it is open source. This makes Sikuli's libraries easy to include in a custom framework.

RoutineBot would be a good alternative to vTask thanks to its cheap license and text recognition. However, there seems to be some compatibility issues with Jenkins and would require some work.

Ranorex is an interesting alternative that combines both GUI testing technologies, image recognition and object recognition. This tool is widely used but the lack of VNC support or

other remote technology is a drawback in this price category. The price is to some extent acceptable as Ranorex has a good support if there are problems.

eggPlant and T-plan are the two enterprise solutions for image based GUI testing with the most functionality and personal support. However, these tools are really interesting only if there will be GUI testing on the LDU hardware with VNC, which is undesired at this time. The rather simple GUI of the LDU may make these tools somewhat overqualified for the task. Therefore it would be interesting to know if a cheap, simple and easy to learn tool is enough to do satisfactory testing.

4 Testing with vTask Studio

vTask Studio or just vTask for short, is an image based testing tool developed by Vista Software for over a decade. The development of vTask is based on feedback from users who are giving suggestions for new features. The main feature of vTask is the lack of scripting language and the built in compiler that allows creation of standalone executable files of the rest scripts that do not require external libraries or even vTask itself to run. vTask is not limited to testing alone as many users use vTask to create for example a username and password managers, setup scripts for larger systems of computers or training sessions of new system users. /10/

4.1 Basic usage

Making scripts in vTask is a simple task. All available actions are situated and categorized in the action tab on the left side of the window. Scripting is done by clicking and dragging the desired actions from the action tab to the main list. The actions are executed from the top down until the script reaches the last action or an end run action.

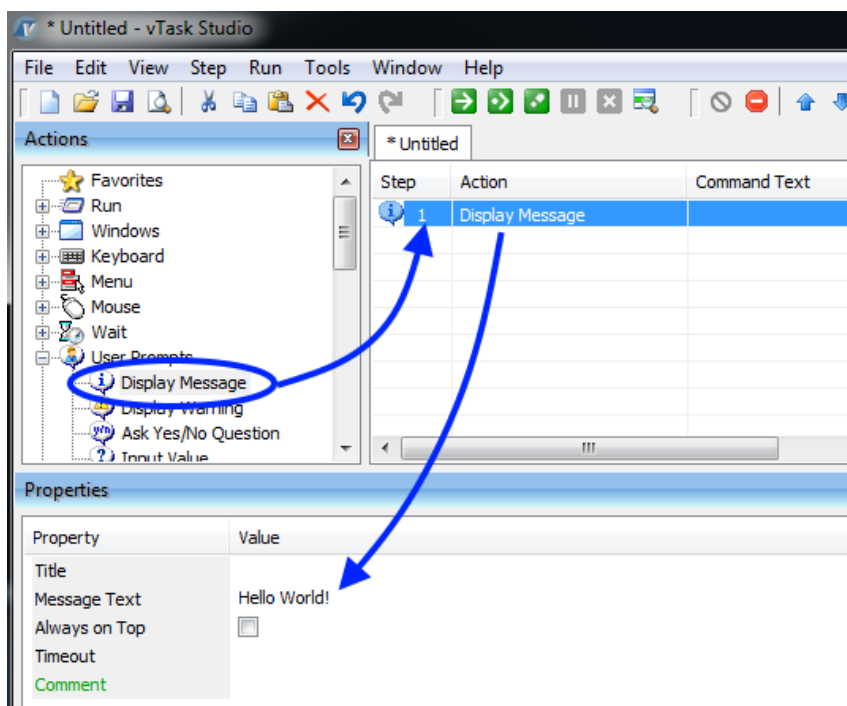


Figure 7. Hellos world example in vTask using actions.

To make a simple “Hello World!” script all that is needed is to click and drag the display message action to the main list. When the action is clicked in the main list, the properties window should now show the properties of the display message action. Depending on which action that is used this window will show different properties to configure. Writing “Hello World!” in the message text property will make an info window that salutes the world.

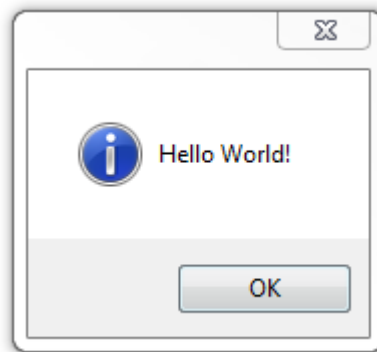


Figure 8. Hello world example.

vTask saves these scripts in an XML format. This makes it easier to share the solution to other vTask users by just copying the XML text. The “Hello World!” script will have the following XML code.

```
<vTask>
<step>
  <action>Display Message</action>
  <text>Hello World!</text>
</step>
</vTask>
```

Figure 9. XML code for “Hello World!” example.

More advanced text that changes depending on a condition can be written with Excel functions. For example, to make a window that shows AM or PM depending on the time can be done by writing with Excel’s IF-function “=IF({hour24}<11, “AM”, “PM”)” in the message text property. The “{hour24}” is a vTask variable that returns the hour part of the system time.

All Excel functions that are supported, available variables and descriptions of the vTask actions can be found in the well written vTask help documentation.

4.2 Compatibility with Jenkins

The most important requirement of the testing tool is compatibility with Jenkins. /11/ Figure 10 represents how Jenkins will be used for the whole GUI testing chain in the future. Jenkins can be scheduled to initiate the testing after a workday, for example at midnight. It all starts with Jenkins checking TestLink for what test cases that need to be run (1.) and then downloading the code that concerns the test cases (2.) ,which is the LDU software in our case. Jenkins then compiles the code to an executable file that simulates the LDU and saves it on the Jenkins server. Jenkins then activates the test scripts (3.) connected to the test cases, which then download the newly compiled LDU software to the test environment and run the tests. After the test cases are finished the test scripts create test results that Jenkins reads (4.) and then make a test report to TestLink (5.).

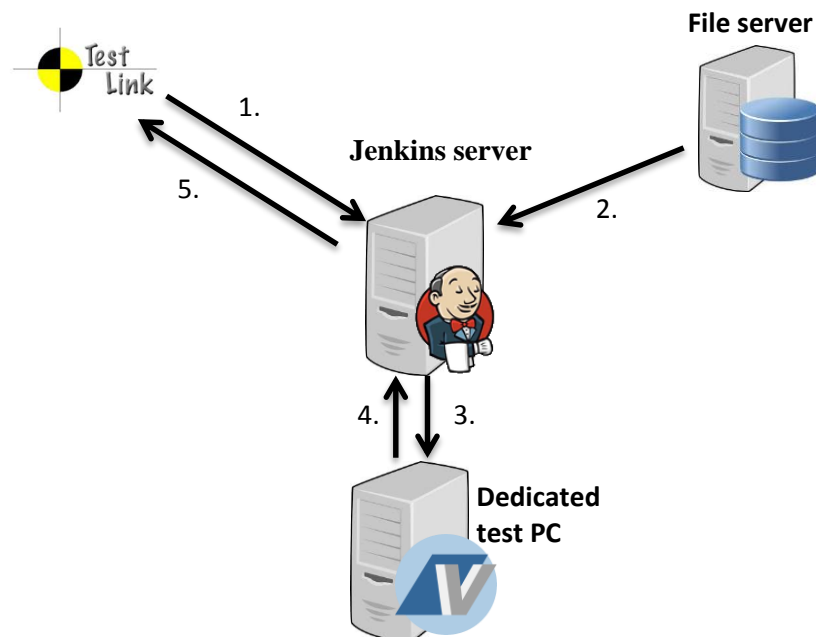


Figure 10. Representation of the testing chain.

To make this system work, Jenkins needs a way to start the test scripts and then receive test results from the executed test cases. The first requirement is met thanks to the possibility to create standalone executable files of the test scripts. Jenkins is able through

the command line to start the test scripts. The second requirement is met through a plugin in Jenkins that accepts TAP-files. TAP-files are a way to communicate test results and other information. They can be written in notepad and then saved with the TAP-file extension. vTask is able to write test results in a TAP file through simple scripting.

TAP-files are written on the following form:

```
1..N
ok 1 - Description
ok 2 - Description
....
ok N - Description
```

Figure 11. TAP-file format.

The first line in a TAP-file will declare how many lines of information there are additionally. If there are three steps in a test case the first line must declare "1..3". The other three lines will then declare if a step was successful or not by declaring "ok" or "not ok" and the step number and then an optional description such as "All buttons found". An example of a test result is found in chapter 4.4.4.

The practical implementation of the whole testing chain will not be included in this thesis because of lack of time to configure Jenkins and TestLink.

4.3 Designing of test cases

To make a proof of concept, two examples of test cases were designed. The first test cases will check the layout of the home page. The home page is the main page of the LDU. The other test case will test the shortcut functionality. Additionally, there will be a functional script that initiates testing. The initiate testing script will start the LDU simulation if it's not already running and go to the home page, which is the base state. Later, this script can also include downloading newest build of the LDU simulation.

4.3.1 Home page layout test

This test case will check vTask ability to do a simple layout test. This test case will check that all buttons and gauges that are supposed to be on the screen are visible.

The home screen consists of 14 different page buttons. The page buttons will bring the user to different pages of the GUI. On these pages there are different functionality and monitoring alternatives. For example the exhaust button will bring the user to a page that shows exhaust gas temperatures. Another button, such as the Log button, will bring the user to a page where system events are registered. On the home screen there are also gauges that show the value of the most important values such as engine speed.

The test script will run the initiate testing script. Then it will one at a time search for the buttons and report if there are errors. Afterwards the script will search for the gauges. The test result will be written to a TAP file.

4.3.2 Shortcut functionality test

This test case will check vTask ability to do some functional testing. The test case will check the GUI behavior when using the shortcut functionality.

The shortcut functionality is a feature that makes it possible to change for example from the exhaust page to the cooling system page without the need to use the home page. The user is able to add five shortcuts to a page. The user should not be able to add the page that is displayed as a shortcut. When a shortcut is selected it should appear at the bottom of the screen and the selected button should be darkened in the shortcut window. When one of the shortcuts is clicked the page should change according to the shortcut.

The test script will run the initiate testing script. Then it will add five shortcuts and check if they are in the shortcut bar and that the buttons in the shortcut window are darkened. The script will try to add a sixth shortcut, which should not appear in the shortcut bar. The script will remove one shortcut and try adding the opened page, which should not be possible. To check if shortcuts are working, a shortcut is clicked, which should open the correct window. The script will then remove all shortcuts and return to home page. The test result will then be written to a TAP file.

4.4 Explanation of scripts

To implement the two test cases, three scripts were created. There are one script each for the test cases and one script to initiate testing.

4.4.1 Test initiation script

The point of this script is to use and test the “*Call script*” action in vTask. The first segment in this script is to make the LDU simulation window visible. This is done by the action “*IF Window Exists*” that is configured to run the LDU simulation’s EXE file if it’s not running and wait for it to start before it declares the window as used or in other words selected. When using other actions such as “*Wait for Window*” or “*Click on Image*” the selection makes it possible to limit the searching to the last used window, which speeds up image searching. If the simulation is started the script will just declare the window as used, which also brings it to the front and restores the window if minimized. If any of the actions in this script fails, they will increase the error counter, initiated in the beginning, by one.

The screenshot shows a script editor with the following content:

```

Script for initiating testing.
Starts the LDU simulation if it 's not running, selects the window and goes to the LDU homepage
#####
Variables
6 Set Variable %Errors = 0 %Errors Reset Errors
Startup script
8 IF NOT Window Exists LDUsim Exact Title,...
9 Start C:\Testing\LDUSimulation\LDUsi... Single Starts the Simulation
10 Wait for Window LDUsim 10 Exact Title Waits for LDU simulation to start
11 Use Window LDUsim 3 Partial Title... Select window to speed up image...
12 ELSE
13 Use Window LDUsim 3 Partial Title... Select window to speed up image...
14 Delay 1 Wait for window to appear if mimi...
15 END IF

```

Line	Action	Target	Count	Criteria	Notes
6	Set Variable	%Errors = 0		%Errors	Reset Errors
8	IF NOT Window Exists	LDUsim		Exact Title,...	
9	Start	C:\Testing\LDUSimulation\LDUsi...		Single	Starts the Simulation
10	Wait for Window	LDUsim	10	Exact Title	Waits for LDU simulation to start
11	Use Window	LDUsim	3	Partial Title...	Select window to speed up image...
12	ELSE				
13	Use Window	LDUsim	3	Partial Title...	Select window to speed up image...
14	Delay		1		Wait for window to appear if mimi...
15	END IF				

Figure 12. Actions to make LDU simulation visible.

The second segment of the script is to make sure the home page is displayed, which is the base state. The “*IF Image Visible*” action will here declare which state the simulation is in. If the simulation is on another page than the home page, the home page button will be visible. By clicking the home button, which is done by using the “*Click on Image*” action, the simulation is brought to the base state. Otherwise, if the shortcut window is open it will find the clicked shortcut button, which is darkened when the shortcut window is open. The script will then click the shortcut button to close the shortcut window and then the home button to bring the simulation to the base state. The script will then write the

result of the initiation to a text variable that is used in the test case results to make tracking for problems easier.

Go to home screen					
17	IF Image is Visible	C:\Testing\Task pics\Home.bmp		ScanLastWin	Home button only visible when not on the ham...
18	Click on Image	C:\Testing\Task pics\Home.bmp	5	ScanLastWin, Snap to Position	Go to home page
19	ELSE IF Image is Visible	C:\Testing\Task pics\SelectedShortcut...		ScanLastWin, ELSE IF	Darkened shortcut button only visible when sh...
20	Click on Image	C:\Testing\Task pics\SelectedShor...	5	ScanLastWin, Snap to Position	Exit shortcut window
21	Click on Image	C:\Testing\Task pics\Home.bmp	5	ScanLastWin, Snap to Position	Go to home page
22	ELSE				Should be in screensaver mode
23	Move to Window			Most Recent "Use Window" ,...	
24	Mouse Click			Left,Down+Up	Clicking in the middle of window to exit screens...
25	END IF				
Reporting					
27	Set Variable	%StartupReport =IF(%Errors=0, "ok ...			

Figure 13. Actions to go to the homes page.

4.4.2 Home page layout script

The first segment of the script for the home page layout test will first delete all variables to make sure there are no conflicting values for some unexpected reason. The script will then call the initiate testing script explained in the previous chapter. The script will then reset errors if there were any in the initiation script. The script will then loop through all buttons available on the home page. The search is limited to the LDU screen. The “*Wait for Image*” action, which waits for an image to appear, is the simplest way to check if the buttons are visible. If a button isn’t found within the time set in the timeout property, the check will fail and increment the error variable. The script will then write the results of the second step to a variable.

Layout test for home page					
Checks if all buttons and gauges are visible on home page.					
#####					
Initiate testing					
6	Delete Variable			All	
7	Call Script	C:\Testing\vTask scripts\InitiateTesting...		Wait For Exit	<i>Calls the initiate testing script</i>
Variables					
9	Set Variable	%Errors = 0			<i>Reset errors</i>
Checking buttons					
11	Wait for Image	C:\Testing\vTask pics\Cooling.bmp	1	ScanLastWin	
12	Wait for Image	C:\Testing\vTask pics\Crank.bmp	1	ScanLastWin	
13	Wait for Image	C:\Testing\vTask pics\ChargeAir.bmp	1	ScanLastWin	
14	Wait for Image	C:\Testing\vTask pics\Exhaust.bmp	1	ScanLastWin	
15	Wait for Image	C:\Testing\vTask pics\ExhaustDeviation...	1	ScanLastWin	
16	Wait for Image	C:\Testing\vTask pics\Ignition.bmp	1	ScanLastWin	
17	Wait for Image	C:\Testing\vTask pics\Info.bmp	1	ScanLastWin	
18	Wait for Image	C:\Testing\vTask pics\Knock.bmp	1	ScanLastWin	
19	Wait for Image	C:\Testing\vTask pics\Log.bmp	1	ScanLastWin	
20	Wait for Image	C:\Testing\vTask pics\Lube.bmp	1	ScanLastWin	
21	Wait for Image	C:\Testing\vTask pics\Misc.bmp	1	ScanLastWin	
22	Wait for Image	C:\Testing\vTask pics\Pressure.bmp	1	ScanLastWin	
23	Wait for Image	C:\Testing\vTask pics\Temperature.bmp	1	ScanLastWin	
24	Wait for Image	C:\Testing\vTask pics\Gas.bmp	1	ScanLastWin	
25	Set Variable	%ButtonReport =IF(%Errors=0, "ok 2 ...			
26	Set Variable	%Errors = 0			<i>Reset errors</i>

Figure 14. Actions for test initiation and button check.

The second segment of the script will in the same way as the buttons search for the gauges that should be on the home page. This should be more taxing than the images of the buttons as the gauges are larger, which means more pixels to compare. Then the script will write results of the third step to a variable and create the TAP-file of the results for the test case.

Checking gauges					
28	Wait for Image	C:\Testing\vTask pics\GaugeEngineSpe...	1	ScanLastWin	
29	Wait for Image	C:\Testing\vTask pics\GaugeHtWaterTe...	1	ScanLastWin	
30	Wait for Image	C:\Testing\vTask pics\GaugeLoad.bmp	1	ScanLastWin	
31	Wait for Image	C:\Testing\vTask pics\GaugeLoPressure...	1	ScanLastWin	
32	Wait for Image	C:\Testing\vTask pics\FuelDemand.bmp	1	ScanLastWin	
33	Set Variable	%GaugeReport =IF(%Errors=0, "ok 3 ...			<i>Writes results</i>
Test reporting					
35	Write/Create File	C:\Testing\vTask results\HomePageLay...		Erase	<i>Creates TAP-file</i>

Figure 15. Actions for gauge check and TAP-file creation.

4.4.3 Shortcut functionality script

The first segment of the script for the shortcut functionality test case will delete all variables and initiate testing. The error counter is reset and coordinates for the shortcut bar are added. These are used when checking for the added shortcuts that should or should not be visible in the shortcut bar. By using the vTask built in values “{window_x}” and “{window_y}” that return the window position and the values “{window_width}” and

“{window_height}” that return the width and height of the used window, it’s possible to make the testing more robust as the window position doesn’t matter.

The screenshot shows a test script editor with the following content:

```

GUI test for shortcut functionality
Checks that the five added shortcuts are visible in shortcut bar and that the buttons are darkened
when selected in shortcut window. Checks that a sixth shortcut can't be added. Checks that
the used page can't be added as a shortcut. Checks that a shortcut leads to the correct page.
#####
Initiate testing
8 Delete Variable All
9 Label Restart
10 Call Script C:\Testing\vTask scripts\InitiateTesting... Wait For Exit
Variables
12 Set Variable %Errors = 0 Resets Errors
13 Set Variable %ShortcutBarStartY = {window_y}+690 Coordinates for shortcut bar
14 Set Variable %ShortcutBarStartX = {window_x} *
15 Set Variable %ShortcutBarEndY = {window_y}+{win... *
16 Set Variable %ShortcutBarEndX = {window_x}+{win... *

```

8	Delete Variable		All	
9	Label	Restart		
10	Call Script	C:\Testing\vTask scripts\InitiateTesting...	Wait For Exit	
Variables				
12	Set Variable	%Errors = 0		Resets Errors
13	Set Variable	%ShortcutBarStartY = {window_y}+690		Coordinates for shortcut bar
14	Set Variable	%ShortcutBarStartX = {window_x}		*
15	Set Variable	%ShortcutBarEndY = {window_y}+{win...		*
16	Set Variable	%ShortcutBarEndX = {window_x}+{win...		*

Figure 16. Actions for test initiation and setting up variables.

Figure 17 represents steps two to four. The second step will go to the info page and open the shortcut window and add five shortcuts by clicking page buttons and write results to a variable. Before the shortcuts are added a check is run to reset all shortcuts, which is done in the ResetShortcuts function described later. After vTask has tried to add the five shortcuts an error check is done in a function described later. The functions are called by the “GOSUB Label” action that jumps to the specified label situated at the end of the script. If there are any errors when adding the shortcuts, which in this case means that vTask can’t find the buttons, the error check will try to restart the test script. If no errors are found the script continues to write test results to a variable and reset error variable.

The third step will check that the clicked page buttons in the shortcut window are darkened, which means the page has been added as a shortcut. The script will then write the results to a variable and reset the error counter.

The fourth step will check that the clicked page buttons in the shortcut window are added to the shortcut bar. This is done by exiting the shortcut window and limiting the search for each page button to the shortcut window. The script will then write the results to a variable and reset the error counter.

<i>Go to Info page and add five shortcuts</i>						
18	Delay		1			
19	Click on Image	C:\Testing\Task pics\Info.bmp	5	ScanLastWin, Snap to Position		<i>Go to Info page</i>
20	Click on Image	C:\Testing\Task pics\Shortcut.bmp	5	ScanLastWin, Snap to Position		<i>Open shortcut window</i>
21	GOSUB Label	ResetShortcuts				<i>Checks that all shortcuts are reset</i>
22	Click on Image	C:\Testing\Task pics\ChargeAir.bmp	5	ScanLastWin, Snap to Position		<i>Add shortcuts</i>
23	Click on Image	C:\Testing\Task pics\Cooling.bmp	5	ScanLastWin, Snap to Position		*
24	Click on Image	C:\Testing\Task pics\Crank.bmp	5	ScanLastWin, Snap to Position		*
25	Click on Image	C:\Testing\Task pics\Pressure.bmp	5	ScanLastWin, Snap to Position		*
26	Click on Image	C:\Testing\Task pics\Temperature.bmp	5	ScanLastWin, Snap to Position		*
27	GOSUB Label	Error check				<i>Go to error check label</i>
28	Set Variable	%ShortcutAdding =IF(%Errors==0, "o...				<i>For test report</i>
29	Set Variable	%Errors = 0				<i>Resets Errors</i>
<i>Check that the clicked pages are selected (darkened)</i>						
31	Wait for Image	C:\Testing\Task pics\SelectedChargeA...	1	ScanLastWin		
32	Wait for Image	C:\Testing\Task pics\SelectedCooling...	1	ScanLastWin		
33	Wait for Image	C:\Testing\Task pics\SelectedCrank.bmp	1	ScanLastWin		
34	Wait for Image	C:\Testing\Task pics\SelectedPressure...	1	ScanLastWin		
35	Wait for Image	C:\Testing\Task pics\SelectedTempera...	1	ScanLastWin		
36	Set Variable	%ShortcutSelect =IF(%Errors==0, "ok...				<i>For test report</i>
37	Set Variable	%Errors = 0				<i>Resets Errors</i>
<i>Check that added pages are added in the shortcut bar. Search limited to shortcut bar</i>						
39	Click on Image	C:\Testing\Task pics\SelectedShortcut...	5	ScanLastWin, Snap to Position		<i>Close shortcut window</i>
40	Wait for Image	C:\Testing\Task pics\ChargeAir.bmp	1			
41	Wait for Image	C:\Testing\Task pics\Cooling.bmp	1			
42	Wait for Image	C:\Testing\Task pics\Crank.bmp	1			
43	Wait for Image	C:\Testing\Task pics\Pressure.bmp	1			
44	Wait for Image	C:\Testing\Task pics\Temperature.bmp	1			
45	Set Variable	%ShortcutVisible =IF(%Errors==0, "ok...				<i>For test report</i>
46	Set Variable	%Errors = 0				<i>Resets Errors</i>

Figure 17. Actions for adding shortcuts and checking of shortcuts.

The fifth step will check that it is not possible to add a sixth shortcut. This is done by opening the shortcut window and trying to click a sixth page button. Because there is no default action that fails if an image is visible, a manual if-statement is needed that checks if the image is visible and increments the error variable if it's found. The if-statement tries to find the darkened page button, which should fail and continue to close the shortcut window. In the same way the sixth page button is searched for in the shortcut bar, which should fail. The script will then write the results to a variable and reset the error counter.

<i>Try adding a sixth shortcut, check if darkened and check shortcut bar. Should not be darkened and shortcut not added</i>						
48	Click on Image	C:\Testing\Task pics\Shortcut.bmp	5	ScanLastWi...		<i>Open shortcut window</i>
49	Click on Image	C:\Testing\Task pics\ExhaustDevia...	5	ScanLastWi...		<i>Add sixth shortcut</i>
50	IF Image is Visible	C:\Testing\Task pics\SelectedExha...		ScanLastWin		<i>Button should not be found</i>
51	Increment Variable	%Errors++			%Errors	
52	END IF					
53	Click on Image	C:\Testing\Task pics\SelectedShort...	5	ScanLastWi...		<i>Close shortcut window</i>
54	IF Image is Visible	C:\Testing\Task pics\ExhaustDevia...				<i>Button should not be found in sh...</i>
55	Increment Variable	%Errors++			%Errors	
56	END IF					
57	Set Variable	%SixtShortcut =IF(%Errors==0, "o...			%SixtS...	<i>For test report</i>
58	Set Variable	%Errors = 0			%Errors	<i>Resets Errors</i>

Figure 18. Actions for trying to add sixth shortcut that should not be possible.

The sixth step will try to add the info page, which is the opened page, to the shortcut bar, which should not be possible. This is done by again opening the shortcut window and

removing one shortcut by clicking a darkened page button. The script will then click the info page button and search for the darkened info page button, which should not be found. The info button should also not be found in the shortcut bar. The script will then write the results to a variable and reset the error counter.














 <i>Remove a shortcut and try adding the opened page as a shortcut and check if darkened and check shortcut bar. Should not be darkened and shortcut not added</i>						
	61	Click on Image	C:\Testing\vTask pics\Shortcut.bmp	5	ScanLastWin, Snap to Position	<i>Open shortcut window</i>
	62	Click on Image	C:\Testing\vTask pics\SelectedChargeA...	5	ScanLastWin, Snap to Position	<i>Remove shortcut</i>
	63	Click on Image	C:\Testing\vTask pics\Info.bmp	5	ScanLastWin, Snap to Position	<i>Try to add open page</i>
	64	IF Image is Visible	C:\Testing\vTask pics\SelectedInfo.bmp		ScanLastWin	<i>Button should not be found</i>
	65	Increment Variable	%Errors++			
	66	END IF				
	67	Click on Image	C:\Testing\vTask pics\SelectedShortcut...	5	ScanLastWin, Snap to Position	<i>Close shortcut window</i>
	68	IF Image is Visible	C:\Testing\vTask pics\Info.bmp			<i>Button should not be found in shortcut bar</i>
	69	Increment Variable	%Errors++			
	70	END IF				
	71	Set Variable	%OpenPageShortcut =IF(%Errors==0...			<i>For test report</i>
	72	Set Variable	%Errors = 0			<i>Resets Errors</i>

Figure 19. Actions for adding the opened page as shortcut, which should not be possible.

The seventh step will check if clicking a shortcut in the shortcut bar will change to the selected page. This is done, in this case, by clicking the cooling page shortcut. To check if the correct page is showing the title of the page is searched. This should only be visible on the cooling page. The script will then write the results to a variable and reset the error counter.






 <i>Try clicking a shortcut to check if correct page is shown</i>						
	74	Click on Image	C:\Testing\vTask pics\Cooling.bmp	5	ScanLastWin, Snap to Position	<i>Open Page</i>
	75	Wait for Image	C:\Testing\vTask pics\CoolingPage.bmp	1	ScanLastWin	
	76	Set Variable	%UseShortcut =IF(%Errors==0, "ok 7 ...			<i>Resets Errors</i>
	77	Set Variable	%Errors = 0			<i>Resets Errors</i>

Figure 20. Actions to check if shortcuts are working.

The eighth step is a reset step that goes back to the info page and opens the shortcut window and uses the ResetShortcuts function to remove all shortcuts. The script will then close the shortcut window and return to the base state, the home page. The script will then write the results to a variable. The TAP-file is created and the test ends with the “Exit Run” action.

Reset shortcuts and return to home page					
79	Click on Image	C:\Testing\vTask pics\Home.bmp	5	ScanLastWin,Snap to Position	Go to home page
80	Delay		1		
81	Click on Image	C:\Testing\vTask pics\Info.bmp	5	ScanLastWin,Snap to Position	Go to Info page
82	Click on Image	C:\Testing\vTask pics\Shortcut.bmp	5	ScanLastWin,Snap to Position	Open shortcut window
83	GOSUB Label	ResetShortcuts			
84	Click on Image	C:\Testing\vTask pics\SelectedShortcut...	5	ScanLastWin,Snap to Position	Close shortcut window
85	Click on Image	C:\Testing\vTask pics\Home.bmp	5	ScanLastWin,Snap to Position	Go to home page
86	Set Variable	%Reset =IF(%Errors==0, "ok 8 - Rese...			For test report
Test Reporting					
88	Write/Create File	C:\Testing\vTask results\ShortcutFunci...		Erase	Making test report
89	Exit Run				Exit test

Figure 21. Actions for resetting the shortcuts, returning to base state and test result writing.

Figure 22 represents the two functions used in this script. These are called by the “GOTO Label” or “GOSUB Label” actions. There is no direct function call but instead labels or flags are used. The first function is the error check. This function will check if there are errors. If there are no errors the script will jump back to the step it came from by using the “Return from GOSUB”. If there are errors and no restarts have been done, the function will increment the restart variable and go to the restart label in the beginning of the script by using the “GOTO Label”. If there are errors and a restart had been done, the script will automatically fail all steps because of an unexpected error and create a TAP-file before exiting the run. This is an example to make the test scripts more robust if there are unexpected errors. This is not good as a versatile solution for all situations, but with a little bit more development the test scripts can be very robust.

The last function is the ResetShortcuts function. This function will search for darkened buttons in the shortcut window. To avoid search each darkened page button individually, the script will loop a search for the borders of the buttons and click it until there are no darkened borders to be found. This is done by masking the image. vTask is instructed to ignore all pixels with the magenta color, which has the color code of 0xFF00FF. By using an image editor, the middle portion of the buttons that contain the name of the button can be ignored and thereby makes the resetting of shortcuts easier and faster. The most important thing when making the masked images is to not use any anti-aliasing when painting over the ignored areas. The anti-aliasing will try to smooth out the magenta and dark color at the border, which will make some pixels that should be magenta to be somewhat darker and will be added to pixels to match for the search logic. When no more darkened buttons can be found, the function will return to the script.

Functions					

Error check. This is one example on how to make the test script more robust					
94	Label	Error check			
95	IF Expression	%Errors==0			If there are no errors return to script
96	Return from GOSUB				
97	ELSE IF Expression	%Errors>0 %Restarts==0	ELSE IF		If there are errors, try to reset
98	Increment Variable	%Restarts++			
99	GOTO Label	Restart			
100	ELSE IF Expression	%Errors>0 %Restarts>0	ELSE IF		If there are errors and a restart has been done
101	Write/Create File	C:\Testing\vTask results\ShortcutF...	Erase		Fail all tests
102	Exit Run				Exit test
103	END IF				
Reset shortcuts. Function to reset shortcuts					
105	Label	ResetShortcuts			
106	Start 'While' Loop	1			Loops until all shorcuts are removed
107	IF Image is Visible	C:\Testing\vTask pics\ResetSelecte...	Masked,ScanLastWin		If a shortcut is added, click to remove
108	Click on Image	C:\Testing\vTask pics\ResetSele...	5	Masked,ScanLastWin, Snap ...	
109	ELSE				
110	Break out of Loop				
111	END IF				
112	NEXT LOOP				
113	Return from GOSUB				Return to script

Figure 22. Actions for the two functions used in the script.

4.4.4 Test result TAP-files

The following figure represents the resulting TAP-file that is created by the home page layout script. This is just an example as the final testing chain may just want one step in each TAP-file. For example the shortcut functionality script would make eight TAP-files instead of having 8 steps in a single TAP-file.

<pre> 1..3 ok 1 - Simulation started successfully ok 2 - All buttons visible ok 3 - All gauges visible </pre>	<pre> 1..3 not ok 1 - FAILED: Failed to initiate testing not ok 2 - FAILED: 14 button(s) not visible not ok 3 - FAILED: 5 gauges(s) not visible </pre>
---	---

Figure 23. Example of successful test (left) and a failed test (right).

5 Results

Overall, vTask was a pleasant tool to use for this test environment and purpose. The test scripts behaved as planned and were able to find or click anything that it was instructed to do. The scripts were compiled into executable files and were run from the command line, which functioned correctly as when executed from vTask. The TAP-files were correctly created with the correct content and saved with the correct file extension. The test scripts were reasonably quick in their test execution. The shortcut functionality test case took only 38 seconds to complete for the test script. This is quite fast compared to a user familiar to the LDU that took 95 seconds to complete the test and an unfamiliar user took 276 seconds to complete the test.

The positive sides of vTask are:

- + **Ease of use.** The tool is in its basics very simple to use. The action based programming is simple, smooth and customizable. The scripts are quite easy to get an overview of and to understand. Needed math and logical calculations and data editing is done with excel-functions, which most engineers are used to.
- + **Little time needed to make test scripts.** Making scripts in vTask was surprisingly efficient. As soon as some basic assisting scripts have been made to make testing more robust, such as restarting the simulation if it crashes, the test case scripting should be fairly quick.
- + **Good help documentation.** The documentation is easy to understand and usually includes examples on how to use the different actions and variables. The online vTask forum is also a good place to see different solutions to problems that can occur.
- + **Good maintainability.** Maintainability is fairly good in vTask because of the image based testing method, which makes the position of the object irrelevant. If there are changes for example in button design the images of the buttons used by vTask will unfortunately need to be updated. However, taking new screenshots of the buttons will not take a long time. Nothing has to be done to the script itself as it will automatically start using the updated screenshots.

- + **Can be robust.** The test scripts can be very robust. However it will require time to make all the custom scripts that will increase robustness. Nevertheless, as the example function, used in the shortcut functionality test case, vTask proves that it is capable of making robust test scripts.
- + **All variables global.** All created variables are on a global level, which is good when the developer don't have to worry about the functions or external scripts ability to read the values of the variables. If a variable is set to value in an external script the main script can use it directly without the need to make the external script return a value to the main script.
- + **Test reporting fairly easy.** The ability to create files with any file extension makes the TAP-file creation very easy. To declare a test case failed or successful is easy, but more specific descriptions of what failed in addition to the amount of errors, will require more time to develop automatic custom scripts.
- + **Compiled scripts works well.** vTask is able to compile the scripts to an executable file that works completely independent by adding all the pictures used in the script to the executable file. This will only require the external scripts to be on the same computer. There is also the ability to encrypt the scripts if needed.
- + **Limiting search area.** The ability to cut down the search area decreases the time needed to execute the tests. The ability to only search last used window is a good start. If there is a requirement to limit the search area to a specific part of the screen it's possible as seen in the shortcut functional test case.

There are unfortunately some negative sides working with vTask, which are:

- ***“Wait for Image”* action slow.** There seems to be some inertia in this action. The *“Click on Image”* and *“If Image is Visible”* seems to find the image faster. The explanation to the inertia may be that it is on purpose slower to make sure the wait is long enough for next action, which depends on the wait action, to be able to do its job.
- **Variables needed to be made to limit search area on a window.** The start and end coordinates property to limit search area doesn't support calculations. The coordinates must be calculated in variables before adding it to the action

properties. This is a minor incontinence as it may actually be preferable to calculate in the variables first as they can be initiated in the beginning and be used for many actions. If the search area must change, all that is needed is to update the coordinates in the beginning, which is good for maintainability.

- **Occasionally too fast “Click on Image” action.** This action will occasionally find and click buttons before they are clickable. Apparently there is a short time in the animation when the page changes where vTask manage to find the searched image.
- **Can’t queue several “On Failure”.** There is a feature that can be activated in properties for most actions that makes it possible to do something when the action fails. In the test scripts the “*Increment Variable*” alternative is used to keep track on the number of errors. There are other interesting alternatives to choose from, such as “*Retry Step x Times*” and “*Jump to Label*”. However, only one can be added to each action. On the other hand, this is a feature that can be requested as the developers of vTask are basing their development from user feedback. The workaround at the moment is to make special functions to take care of the script robustness.
- **All variables global.** To have all variables global also have some inconveniences. The naming of variables is more important so that values are not overwritten unintentionally. For example if the main script has a variable named “*Errors*” and calls an external vTask script, which has an error variable with the same name. This variable is probably reset in the beginning of the external script to keep track on the errors, which will then overwrite the variable used in the main script. It is important to keep track of the names of the used variables.

All in all the proof of concept was a success. vTask fulfills our expectations and is a worthy candidate for the future development of the automatic GUI testing.

6 Discussion

All in all using vTask to develop GUI testing for the LDU was a good experience. With all the information, all pros and cons, in this thesis I think it is enough to attempt a more full scale development. If the requirements stay as it was at the time this thesis was written, vTask will be up to the challenge to take care of the GUI testing for the LDU. However, this thesis describes just one part of the whole testing system. The LDU simulation will need some work before it really can match the complete LDU functionality. This version of the LDU simulation has no updating of values and pages are pictures except one concept page. The next step in the automatic GUI testing development would be to get the simulation working properly as it is a showstopper at the moment. Jenkins will also need configuring to start the real testing. Last but not least, a great deal of test cases must be designed according to the features and requirements of the LDU.

If there had been more time it would have been interesting to make a proof of concept of all the tools listed in this thesis and try out the whole process for the testing with Jenkins and TestLink. This would make the conclusion more bulletproof. The choice of tool to use for the proof of concept was based on how they compared on paper. This is quite a limitation. Just because vTask on paper was the most attractive alternative doesn't automatically mean it is the best. There are still some factors like usability and efficiency, which are very important and unfortunately difficult to represent by words and numbers.

As for me, this thesis was very interesting to write. I have learned a lot about testing, such as why it is done, how to structure testing and how to make good automatic testing. I have also learned to be open to changes. vTask that was finally chosen, was just one of many tools that I had already in advance concluded in my mind to be the tool to use. This also challenged me to be more unbiased and objective, which can be hard when there is already a solution to something.

7 List of Sources

/1/ *Wärtsilä overview* (n.d.).

<http://www.wartsila.com/en/about/company-management/overview> (retrieved: 28.8.2014).

/2/ *Wärtsilä logo* (n.d.).

<http://www.wartsila.com/en/about/company-management/strategy/brand> (retrieved: 28.8.2014)

/3/ Li, K. & Wu, M. (2008). *Effective GUI Test Automation: Developing an Automated GUI Testing Tool*. Alameda: SYBEX.

/4/ Oshero, R. (2009). *The Art of Unit Testing with Examples in .NET*. Greenwich: Manning

/5/ Agarwal, B.B., Tayal, S.P. & Gupta, M. (2010). *Software engineering and testing*. Sudbury: Jones and Bartlett Publishers.

/6/ Hayes, L.G. (2004). *The Automated Testing Handbook*. (2. ed.) Richardson: Software Testing Institute.

/7/ Fewster, M. & Graham, D. (1999). *Software Test Automation: Effective use of test execution tools*. Harlow: Addison-Wesley.

/8/ LDU Development Group. (2014). *LDU concept*. Internal document.

/9/ Pes, R. (2009). *Image Based Versus Object Oriented Testing*.

http://www.t-plan.com/robot/docs/articles/img_based_testing.html (retrieved: 9.7.2014).

/10/ *vTask User's Guide*. (n.d.).

<http://www.vtaskstudio.com/help/> (retrieved 14.7.2014).

/11/ *Meet Jenkins*. (2014).

<https://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins> (retrieved 13.8.2014).

/12/ *Sikuli homepage*. (n.d.). <http://www.sikuli.org/> (retrieved 1.7.2014).

/13/ *vTask studio homepage*. (n.d.). <http://www.vtaskstudio.com/> (retrieved 1.7.2014).

/14/ *RoutineBot homepage*. (n.d.). <http://www.routinebot.com/> (retrieved 1.7.2014).

/15/ *Ranorex homepage*. (n.d.). <http://www.ranorex.com/> (retrieved 1.7.2014).

/16/ *eggPlant homepage*. (n.d.).

<http://www.testplant.com/eggplant/> (retrieved 1.7.2014).

/17/ *T-Plan homepage*. (n.d.). <http://www.t-plan.com/> (retrieved 1.7.2014).

Appendix 1

Table 1. Automatic GUI testing tools comparison.

Tool	Scripting language	Test reporting to Jenkins	Command line support	Integrated VNC support	Text recognition	Current version	Upgrades and updates ¹	Price
Sikuli	Python	Yes, scripting needed	Yes	No	No	1.0.1	Free	Free
vTask Studio	Function block scripting	Yes, scripting needed	Yes	No	No	7.87	New licence needed for upgrade. Free updates.	Perpetual 40€
RoutineBot	Pascal, Jscript, Basic	Yes, scripting needed	Yes	No	Yes	3.8	Free upgrades for 1 year, future upgrades have discounted price. Free updates.	Perpetual 500€
Ranorex	C#, VB.NET	Yes, plugin	Yes	No	Yes	5.1.1	Free for 1 year, then subscription based	Perpetual 2000€
eggPlant	SenseTalk	Yes, plugin	Yes	Yes	Yes	14.12	Free	Subscription 6000€
T-Plan	Jscript	Yes, plugin	Yes	Yes	Yes	3.5.2	Free Perpetual licence requires a update subscription 1300€/year	Subscription 3800€ Perpetual 6300€

¹Updates are minor changes e.g. v3.1 to v3.2. Upgrades are major changes e.g. v3 to v4.

This table is a short comparison between the testing tools taken into consideration in this thesis. /12//13//14//15//16//17/