Aalto University
School of Electrical Engineering
Degree Programme in Automation and Systems Technology

Sakari A. Pesonen

# An open and general CNC and machine vision based architecture for payment terminal acceptance test automation

Master's Thesis
Espoo, May 13, 2016

**DRAFT! — May 6, 2016 — DRAFT!**

Supervisor: D.Sc. Seppo Sierla, Aalto University
Advisor: Tatu Kairi M.Sc.

Aalto University
School of Electrical Engineering
Degree Programme in Automation and Systems Technology

ABSTRACT OF
MASTER'S THESIS

| | | | |
|---|---|---|---|
| **Author:** | Sakari A. Pesonen | | |
| **Title:** | An open and general CNC and machine vision based architecture for payment terminal acceptance test automation | | |
| **Date:** | May 13, 2016 | **Pages:** | vi + 14 |
| **Major:** | Intelligent Products | **Code:** | T-110 |
| **Supervisor:** | D.Sc. Seppo Sierla | | |
| **Advisor:** | Tatu Kairi M.Sc. | | |

A dissertation or thesis is a document submitted in support of candidature for a degree or professional qualification presenting the author's research and findings. In some countries/universities, the word thesis or a cognate is used as part of a bachelor's or master's course, while dissertation is normally applied to a doctorate, whilst, in others, the reverse is true.

!FIXME **Abstract text goes here (and this is an example how to use fixme).** FIXME! Fixme is a command that helps you identify parts of your thesis that still require some work. When compiled in the custom `mydraft` mode, text parts tagged with fixmes are shown in bold and with fixme tags around them. When compiled in normal mode, the fixme-tagged text is shown normally (without special formatting). The draft mode also causes the "Draft" text to appear on the front page, alongside with the document compilation date. The custom `mydraft` mode is selected by the `mydraft` option given for the package `aalto-thesis`, near the top of the `thesis-example.tex` file.

The thesis example file (`thesis-example.tex`), all the chapter content files (`1introduction.tex` and so on), and the Aalto style file (`aalto-thesis.sty`) are commented with explanations on how the Aalto thesis works. The files also contain some examples on how to customize various details of the thesis layout, and of course the example text works as an example in itself.

| | |
|---|---|
| **Keywords:** | ocean, sea, marine, ocean mammal, marine mammal, whales, cetaceans, dolphins, porpoises |
| **Language:** | English |

Aalto-yliopisto
Sähtkötekniikan korkeakoulu
Automaatio- ja systeemitekniikan koulutusohjelma

DIPLOMITYÖN
TIIVISTELMÄ

| | | | |
|---|---|---|---|
| **Tekijä:** | Sakari A. Pesonen | | |
| **Työn nimi:** | Ohjelmistoprosessit mänteille | | |
| **Päiväys:** | 13. toukokuuta 2016 | **Sivumäärä:** | vi + 14 |
| **Pääaine:** | Älykkäät tuotteet | **Koodi:** | T-110 |
| **Valvoja:** | D.Sc. Seppo Sierla | | |
| **Ohjaaja:** | Filosofian maisteri Tatu Kairi | | |

Kivi on materiaali, joka muodostuu mineraaleista ja luokitellaan mineraalisisältönsä mukaan. Kivet luokitellaan yleensä ne muodostaneiden prosessien mukaan magmakiviin, sedimenttikiviin ja metamorfisiin kiviin. Magmakivet ovat muodostuneet kiteytyneestä magmasta, sedimenttikivet vanhempien kivilajien rapautuessa ja muodostaessa iskostuneita yhdisteitä, metamorfiset kivet taas kun magma- ja sedimenttikivet joutuvat syvällä maan kuoressa lämpötilan ja kovan paineen alaiseksi.

Kivi on epäorgaaninen eli elottoman luonnon aine, mikä tarkoittaa ettei se sisällä hiiltä tai muita elollisen orgaanisen luonnon aineita. Niinpä kivestä tehdyt esineet säilyvät maaperässä tuhansien vuosien ajan mätänemättä. Kun orgaaninen materiaali jättää jälkensä kiveen, tulos tunnetaan nimellä fossiili.

Suomen peruskallio on suurimmaksi osaksi graniittia, gneissiä ja Kaakkois-Suomessa rapakiveä

| | |
|---|---|
| **Asiasanat:** | AEL, aineistot, aitta, akustiikka, Alankomaat, aluerakentaminen, Anttolanhovi, Arcada, ArchiCad, arkki |
| **Kieli:** | Englanti |

# Acknowledgements

I wish to thank my instructor Tatu Kairi and my supervisor Seppo Sierla for their great help and knowledge throughout the writing process of this thesis.

Espoo, May 13, 2016

Sakari A. Pesonen

# Abbreviations and Acronyms

| | |
|---|---|
| ATT | Automated Acceptance Testing |
| UI | User Interface |
| LCD | Liquid Crystal Display |
| BW | Black and White |
| PIN | Personal Identification Number |

# Contents

# Chapter 1

# Introduction

Software testing is a crucial part of modern software development and it is commonly accepted fact that the earlier defects and errors in the software are found, the lower the cost of correcting those will be. Early detection of errors also increases the possibility to correct them properly. (*Myers et al. [2011]*)

Acceptance testing is a process of comparing the developed program to to the initial requirements (*Myers et al. [2011]*). Therefore especially in agile software development, automated acceptance testing (AAT) plays important role as new versions of software are being developed constantly and AAT phase should be executed whenever new features are added. Automation can free valuable human resources from the process (*Haugset and Hanssen [2008]*) and therefore lover the overall cost of the software.

According to *Sommerville [2011]* acceptance testing of a system should be executed at the final production environment, or at least at environment similar to the production environment. System should also be tested with real data rather than with simulated data. When the software being developed is actually embedded software and the production environment is actually real embedded system, in this case payment terminal, the acceptance testing should be executed on actual payment terminal with actually interacting through the user interface (UI) of the machine. This also leads to a situation where concerns pointed out above are actually being emphasized as late detection of defects in embedded software can considerably raise the overall cost of the system (*Ebert and Jones [2009]*).

*Sommerville [2011]* states that it is practically impossible to perfectly replicate the system's working environment and when considering an embedded system, this can can be even harder. Buttons of the device have to be actually pressed and visual changes on the screen of the device has to be observed. In order to automate this, some sort of test environment has to be

implemented that can observe and manipulate the device through physical word, i.e. not simulating the keystrokes nor reading the LCD communication line. Some kind of joint hardware and software solution has to be created and it also has to mimic real human user as realistically as possible.

This seminar work will discuss the theories related to software testing, testing of embedded systems and the problems stated above. Seminar work will present a proposed architecture for automated acceptance testing of payment terminals including the needed hardware and software.

Research presented in this seminar work was carried in co-operation with Eficode Oy and one of the main payment terminal software provider in the Nordic countries.

## 1.1 Problem statements

In order to survey the topic of this work in adequate level, this seminar work will discuss four different problem statements. problem statements are as follows:

1. What are the benefits of using open source software and how can the architecture be designed to maximally exploit these benefits?

2. What are the distinguishing characteristics between different payment terminals that have impact on automated acceptance testing? How can the architecture be designed to adapt the system to different payment terminals with minimal effort?

3. What kinds of test automation approaches exist and which approach is best suited for payment terminal acceptance test automation?

4. How should test keywords used in test suites be defined to make the test suites compact and understandable? How should keywords be defined to make the tests reusable for other types of payment terminals?

## 1.2 Structure of the Thesis

This seminar work will first discuss the theories and literature related to the topic and will then present proposed architecture of automated test environment for payment terminal software acceptance testing. In the first chapter of this seminar work the topic will be introduced, problem statements will be presented and structure of this work will be explained.

Second chapter will cover the literature review of the topic of this seminar work. Each problem statements will have related subsections and individual problem statements will be discussed on those sections. Each subsection will first give introduction on problem statement's point of view and it will be followed by the most relevant references around the topic. Subsections will point out what has been done earlier and how the fundamental aspects of these previous works can be used as a basis for this work.

Third chapter of this seminar work will present the proposed architecture for automated acceptance test environment for payment terminal software based on literature review done on previous chapter. Chapter will present the fundamental parts of hardware and software needed for this kind of environment. This chapter will have diagrams of proposed software architecture as well as fundamental design of needed hardware.

Fourth and the final chapter will conclude the research done on this seminar work and will summarize the benefits obtained by this kind of environment.

# Chapter 2

# Payment terminal acceptance testing

When developing software with agile methodologies for payment terminals, i.e for embedded system, testing is a crucial part of the process. The earlier the defects and errors in the software are detected, the lower the cost and needed effort will be for correcting those (*Myers et al. [2011]*).

Motivation for this research came from payment terminal software provider as they needed cost efficient and simple as possible automated acceptance test environment in order to lower the costs and speed up the acceptance testing phase of their software development.

In order to automate the acceptance testing of the payment terminals, test environment that can manipulate an observe the device through physical world has to be created. In other words, environment has to have some sort of a robot for pressing the buttons, screen of the device has to be observed and all this must be controlled by some kind of combination of software.

Test environment that can be used in acceptance testing of payment terminals has several challenges to tackle and matters related to physical and technical aspects of the payment terminals have to be considered. This chapter will discuss the background of these challenges. Customer also had desire for open source technologies and this chapter will discuss the benefits obtained by using open source software and and hardware in acceptance testing environment for payment terminals. Chapter will also discuss the different approaches for acceptance testing as well as how should the test suites be defined in order to make them understandable and reusable.

## 2.1 Benefits of Open Source solutions

When designing automated acceptance testing environment from scratch, evaluation and availability of different possible components play significant role in terms of development speed and costs. Software components might need some modification to suit the needs of intended environment and same applies also to the hardware parts.

Open source software provides advantage on these matters over closed source products as the source code is easily available (*Morgan and Finnegan [2007]*). As open source software can be accessed free of charge, component can be easily evaluated by trying out whether they work for the purpose or not.

According to *Paulson et al. [2004]* open source projects usually have fewer defects than closed source projects. Defects are found and fixed rapidly as they are reported openly to the open source community.If defect is found during evaluation of the product, it can be also corrected by the user and by doing this the user can contribute to the project. This on the other hand is hardly never possible with closed software.

*Paulson et al. [2004]* also states that open source projects foster more creativity than closed source counterpart. This means that number of functions added over time is higher on open source projects. When using the product in some new field of use, this can be great advantage as user can report desired feature to the community and it can be added relatively quickly if feature is considered needed by the community.

"Open source" hardware on the other hand means that details and plans of the product and parts are commonly available. This allows that parts can be manufactured and modified by anyone with knowledge and skills to suit individual needs. When detailed part descriptions are available multiple manufacturers can fabricate the parts. This creates competition and therefore usually lowers the price of individual hardware parts.

## 2.2 Common characteristics between payment terminals

When designing automated test environment for different kinds of payment terminals, different physical and technical features have to be taken into account. Environment has to be able to manipulate different types of payment terminals and test structure has to be designed to adapt to needs of different software and software versions running on payment terminals.

Majority of payment terminals share some common characteristics as they are made for same purpose: handling card payments. Scope of this thesis is to view those payment terminals that share three main features: keyboard, screen and card slot. Different types of terminals can be observed in Figure 2.1 and Figure 2.2 bellow.

Screens of the payment terminals differ in terms of size, placement and type. Test environment has to take into account different screen placements and it has to support both black and white (BW) and colored displays.

Keyboards of payment terminals share majority of keys together as number keys are needed for entering the PIN code and accept and decline buttons are needed for accepting and canceling the payment. Keyboard layouts, however, differ between different manufacturers and even amongst different models of the same manufacturer.

Location of the chip card slot is usually on the lover edge of the payment terminal or on top of the screen of the payment terminal. Research done within this thesis is limited to those terminals that have the chip card slot at the lower edge of the payment terminal as this simplifies the hardware needed for test environment. This is described more in depth in Chapter 3.2. This study is also limited to only chip card readers and magnetic stripe readers or near field communication (NFC) payments are not addressed.


Figure 2.1: Two examples of payment terminals from different manufacturers. Left image from: `http://www.netskauppa.fi/images/t/24-85-PrimaryImage.image.ashx`


Figure 2.2: Example of a payment terminal which attaches to a smart phone.

## 2.3 Different approaches for test automation

Problem of testing the payment terminal software in automated way can be viewed at different levels. Most abstract division can be seen if the testing is divided into two levels: white box testing and black box testing. White box testing is a method where source code is investigated and test cases are written to test the internal logic of the program. Black box testing on the other hand concentrates only on the inputs and the outputs of the software. Everything between those is not in a field of interest and black box testing only focuses on whether the right input produces the wanted output. (*Myers et al. [2011]*)

*Khan and Khan [2012]* distinguishes these methods from one another clearly by stating that white box testing is a process where full knowledge of source code is needed in order to write the tests. Black box testing is described in a way that only fundamental aspects of the application has to be known and black box testing has no or only little relevance to internal works of the program.

In this way white box testing can be seen to cover unit and integration testing part of the software testing and the black box testing can be seen covering the acceptance testing part of the testing.

As black box testing is based only external exceptions and behavior of the software (*Khan and Khan [2012]*), acceptance testing of payment terminal

software can be seen to follow this methodology.

Therefore acceptance testing of a payment terminal software can be seen as a testing phase where the UI of the device and use cases of the device are tested at the final production level, i.e. through using the real buttons of the device under test and observing that the expected messages can be seen through the screen of the device under test.

## 2.4 Test suite syntax

```
*** Settings ***
Documentation     A test suite with a single test for valid login.
...               This test has a workflow that is created using keywords in
...               the imported resource file.
Resource          resource.txt

*** Test Cases ***
Valid Login
    Open Browser To Login Page
    Input Username    demo
    Input Password    mode
    Submit Credentials
    Welcome Page Should Be Open
    [Teardown]    Close Browser
```

Figure 2.3: Example of a simple test suite. Source: `http://robotframework.org`

# Chapter 3

# Proposed architecture

A problem instance is rarely totally independent of its environment. Most often you need to describe the environment you work in, what limits there are and so on. This is a good place to do that. First we tell you about the LaTeX working environments and then is an example from an thesis written some years ago.

## 3.1 Overview

To create LaTeX documents you need two things: a LaTeX environment for compiling your documents and a text editor for writing them.

## 3.2 Hardware

## 3.3 Software

When you use `pdflatex` to render your thesis, you can include PDF images directly, as shown by Figure 3.2 below.

You can create PDF files out of practically anything. In Windows, you can download PrimoPDF or CutePDF (or some such) and install a printing driver so that you can print directly to PDF files from any application. There are also tools that allow you to upload documents in common file formats and convert them to the PDF format. If you have PS or EPS files, you can use the tools `ps2pdf` or `epspdf` to convert your PS and EPS files to PDF.

Furthermore, most newer editor programs allow you to save directly to the PDF format. For vector editing, you could try Inkscape, which is a new open source WYSIWYG vector editor that allows you to save directly to PDF.

```
*** Settings ***
Documentation    A test suite for testing the machine movement and OCR
Resource         resource.txt
Test Teardown    Run Keywords    Remove Card    Go Home And Close Connection

*** Variables ***
${DEVICE_NUMBER}    1

*** Test Cases ***
Test Invalid Pin Code
    Set Home And Initialize    ${DEVICE_NUMBER}
    Press    red
    Screen Should Contain Text    Terminal ready
    Press    2    0    0    0
    Screen Should Contain Text    20,00
    Press    green
    Screen Should Contain Text    card
    Insert Card
    Press    1    4    5    2
    Press    green
    Screen Should Contain Text    failed
    Remove Card
    Screen Should Contain Text    Terminal ready

*** Keywords ***
Insert Card
    Card In    ${DEVICE_NUMBER}

Remove Card
    Card Out    ${DEVICE_NUMBER}
```

Figure 3.1: Example test case for invalid PIN code test

For graphs, either export/print your graphs from OpenOffice Calc/Microsoft Excel to PDF format, and then add them; or use `gnuplot`, which can create PDF files directly (at least the new versions can). The terminal type is *pdf*, so the first line of your plot file should be something like `set term pdf ...`.

To get the most professional-looking graphics, you can encode them using the TikZ package (TikZ is a frontend for the PGF graphics formatting system). You can create practically any kind of technical images with TikZ, but it has a rather steep learning curve. Locate the manual (`pgfmanual.pdf`) from your LaTeX distribution and check it out. An example of TikZ-generated graphics is shown in Figure 3.3.

Another example of graphics created with TikZ is shown in Figure 3.4. These show how graphs can be drawn and labeled. You can consult the example images and the PGF manual for more examples of what kinds figures you can draw with TikZ.

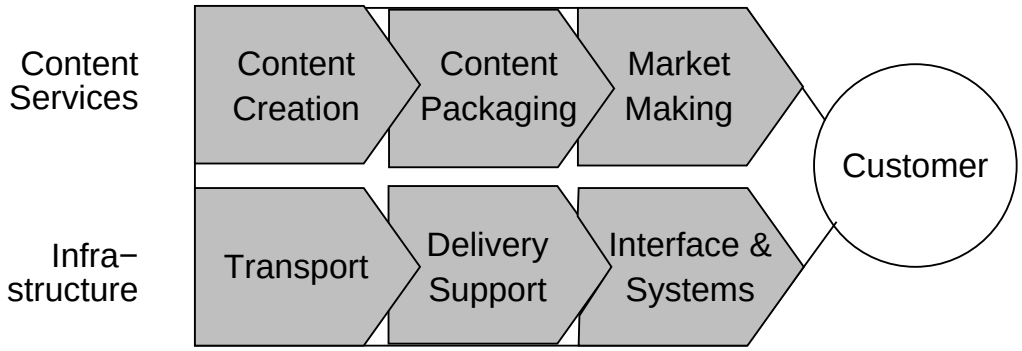
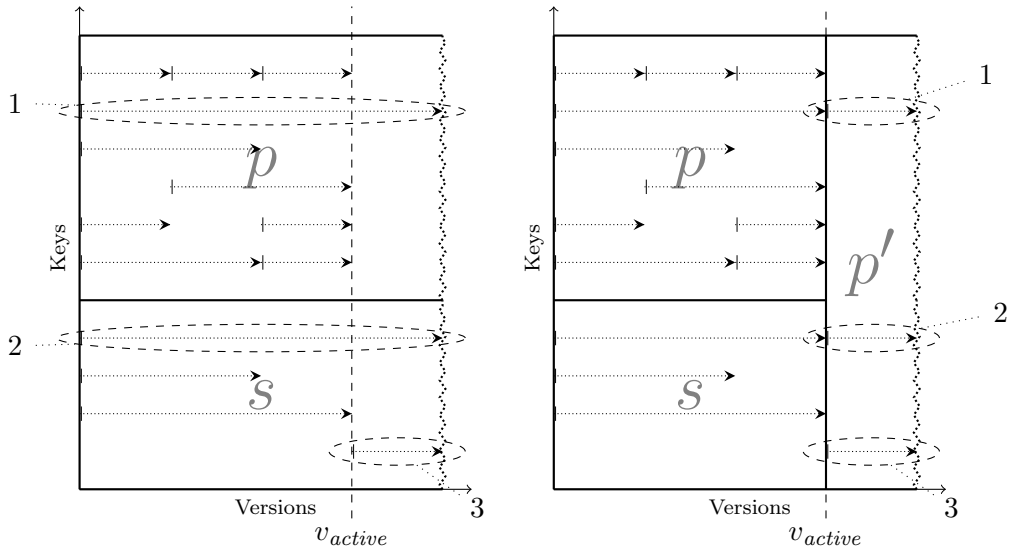
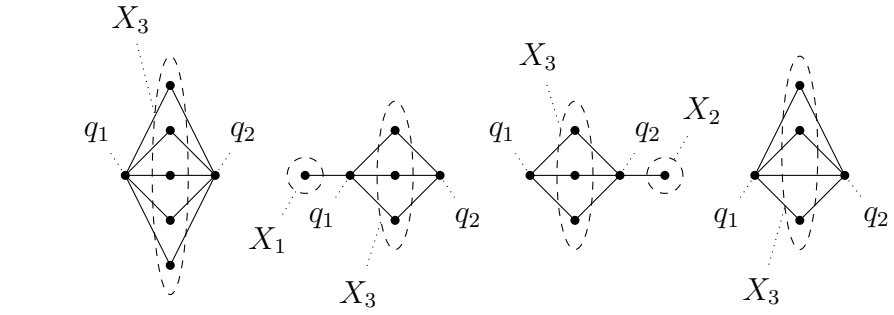



Figure 3.2: The INDICA two-layered value chain model.

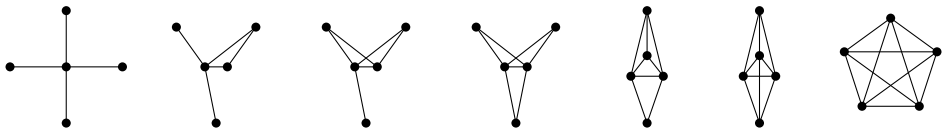



(a) Before (b) After

Figure 3.3: Example of a multiversion database page merge. This figure has been taken from the PhD thesis of.




(a) Examples of obstruction graphs for the Ferry Problem



(b) Examples of star graphs

Figure 3.4: Examples of graphs draw with TikZ. These figures have been taken from a course report for the graph theory course.

# Chapter 4

# Conclusions

Time to wrap it up! Write down the most important findings from your work. Like the introduction, this chapter is not very long. Two to four pages might be a good limit.

# Bibliography


Christof Ebert and Capers Jones. Embedded software: Facts, figures, and future. *Computer*, (4):42–52, 2009.

Borge Haugset and Geir Kjetil Hanssen. Automated acceptance testing: A literature review and an industrial case study. In *Agile, 2008. AGILE'08. Conference*, pages 27–38. IEEE, 2008.

Mohd Ehmer Khan and Farmeena Khan. A comparative study of white box, black box and grey box testing techniques. *Int. J. Adv. Comput. Sci. Appl*, 3(6), 2012.

Lorraine Morgan and Patrick Finnegan. Benefits and drawbacks of open source software: an exploratory study of secondary software firms. In *Open Source Development, Adoption and Innovation*, pages 307–312. Springer, 2007.

G.J. Myers, C. Sandler, and T. Badgett. *The Art of Software Testing*. IT-Pro collection. Wiley, 2011. ISBN 9781118133156. URL `https://books.google.fi/books?id=GjyEFPkMCwcC`.

James W Paulson, Giancarlo Succi, and Armin Eberlein. An empirical study of open-source and closed-source software products. *Software Engineering, IEEE Transactions on*, 30(4):246–256, 2004.

I. Sommerville. *Software Engineering*. International Computer Science Series. Pearson, 2011. ISBN 9780137053469. URL `https://books.google.fi/books?id=l0egcQAACAAJ`.