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 5, 2016 — DRAFT!

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

Aalto University
School of Science

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 + 10
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 + 10
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

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 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

The problem must have some background, otherwise it is not interesting. You can explain the background here. Probably you should change the title to something that describes more the content of this.

2.1 Benefits of Open Source solutions

2.2 Common characteristics between payment terminals

2.3 Different approaches for test automation

2.4 Test suite syntax

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 3.1 below.

You can also include JPEG or PNG files, as shown by Figure 3.2.

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

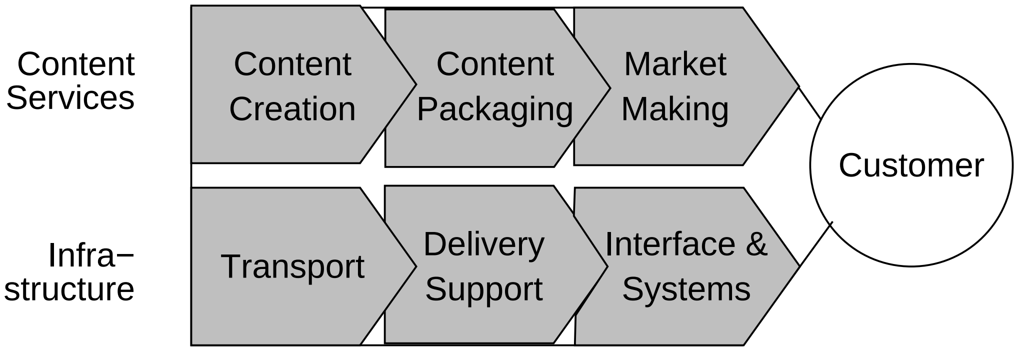


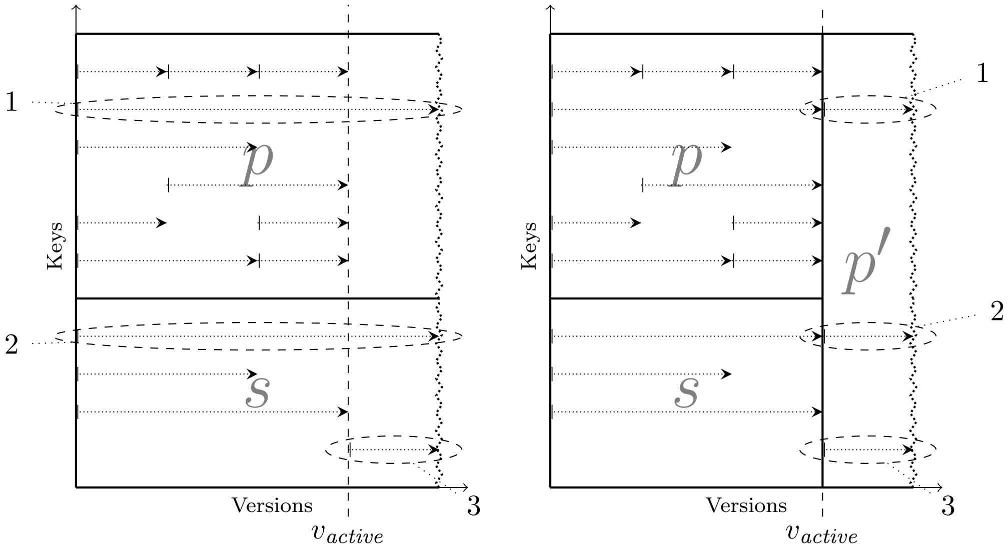
Figure 3.1: The INDICA two-layered value chain model.

source WYSIWYG vector editor that allows you to save directly to PDF. 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.

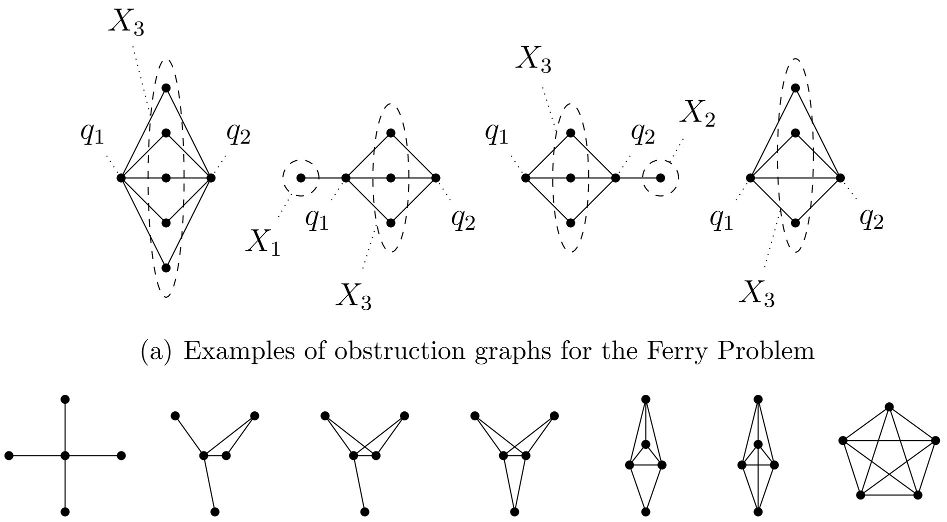
Figure 3.2: Eeyore, or Ihaa, a very sad donkey.



(a) Before

(b) After

Figure 3.3: Example of a multiversion database page merge. This figure has been taken from the PhD thesis of Haapasalo Haapasalo [2010].



(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 Göös et al. [2010].

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.

Mika Göös, Tuukka Haapasalo, and Leo Moisio. Finding hard instances of the ferry problem. Course report for the course T-79.5203/S-72.2420 Graph theory, Aalto SCI, 2010.

Tuukka Haapasalo. *Accessing Multiversion Data in Database Transactions.* PhD thesis, Department of Computer Science and Engineering, Aalto University School of Science and Technology, Espoo, Finland, 2010. `http://lib.tkk.fi/Diss/2010/isbn9789526033600/`.

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.

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`.

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