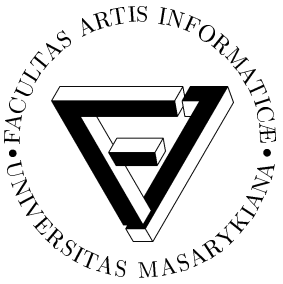
Masaryk University
Faculty of Informatics



Visual testing something catchy

Diploma thesis

Juraj Húska

Brno, 2015

Declaration

Hereby I declare, that this paper is my original authorial work, which I have worked out by my own. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Juraj Húska

Advisor: Mgr. Marek Grác, Ph.D.

Acknowledgement

Some people helped me a lot and some not at all. Nevertheless, I would like to thank all.

Abstract

This thesis is very important!

Keywords

key word1, and so on

Table of contents

1 Introduction

There is a big demand for this thesis. Need and cost of manual testing, space for improvement.

2 Visual testing of software

Testing of software in general is any activity aimed at evaluating an attribute or capability of a program and determining that it meets its required results [1]. It can be done either manually by actual using of an application or automatically by executing testing scripts.

If the application under test has also a graphical user interface (GUI), then one has to verify whether it is not broken. Visual testing of an application is an effort to find out its non-functional errors, which expose themselves by changing a graphical state of the application under test.

Typical example can be a web application, which GUI is programmed usually with combination of HyperText Markup Language (HTML) and Cascading Style Sheets (CSS). HTML is often used to define a content of the web application (such as page contains table, pictures, etc.), while CSS defines a structure and appearance of the web application (such as color of the font, absolute positioning of web page elements, and so on).

The resulting web application is a set of rules (CSS and HTML) applied to a static content (e.g. pictures, videos, text). The combination of rules is crucial, and a minor change can completely change the visual state of the web application. Such changes are very difficult, sometimes even not possible to find out by functional tests of the application. It is because functional tests verify a desired functionality of the web application, and do not consider web page characteristics such as red color of heading, space between two paragraphs, and similar.

That is why a visual testing has to take a place. Again, it is done either manually, when a tester by working with an application, is going through all of its use cases, and verifies, that the application has not broken visually. Or automatically, by e.g. comparing screen captures (also known as screenshots) of new and older versions of the application.

2.1 Visual testing in release testing process

Nowadays software is often released for general availability in repetitive cycles, which are defined according to a particular software development process. Such as Waterfall [2], or Scrum [3].

Testing of software has an immense role in this release process. While automated tests are often executed continuously, as they are quicker to run than manual tests, which are carried out at a specific stage of the release process.

For example in RichFaces[1] Quality Engineering team[2] visual testing was done manually, before releasing the particular version of RichFaces library to a community. In practice it involves building all example applications with new RichFaces libraries, and to go through its use cases with a particular set of web browsers.

1. RichFaces is a component based library for Java Server Faces, owned and developed by Red Hat
2. Quality Engineering team is among the other things responsible for assuring a quality of a product

To be more specific, consider please a web page with a chart elements showing a gross domestic product sector composition in USA (as figure 2.1 demonstrates). Verifying its visual state is not broken, would involve e.g.:

1. Checking the size, overflowing and transparency of all elements in charts.
2. Checking colors, margins between bars.
3. Checking putting of mouse over a specific places in the chart, and verifying whether a popup with more detailed info renders in a correct place.
4. Do this for all major browsers[3], and with all supported application containers[4].

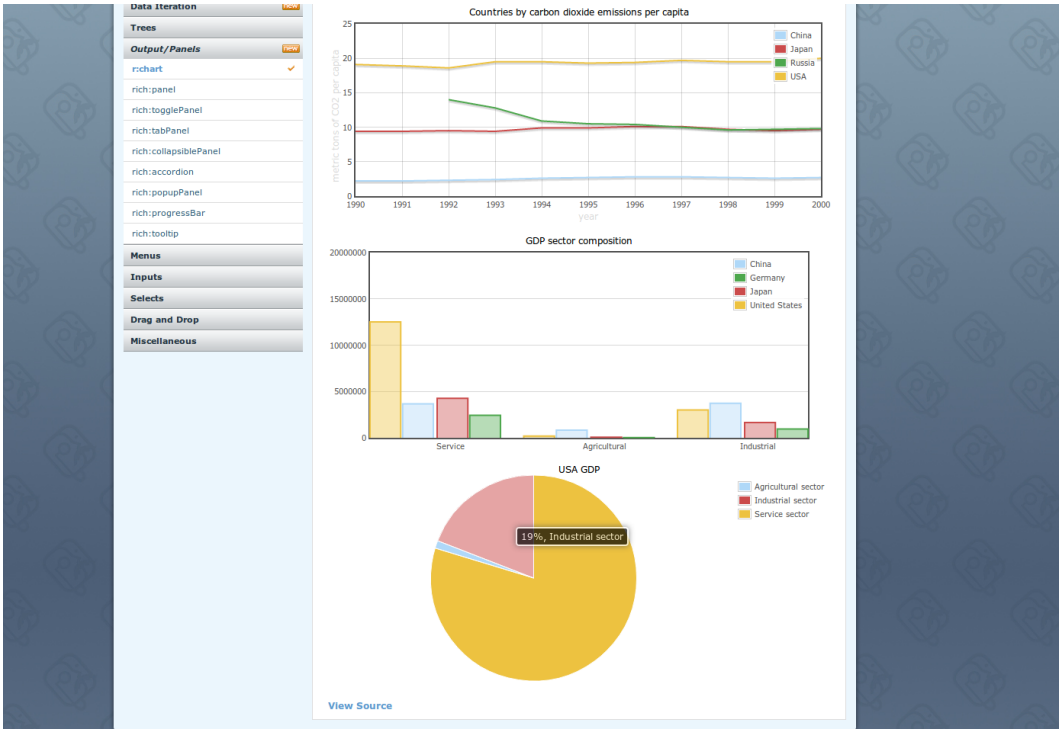


Figure 2.1: RichFaces chart component shown in Showcase application

3. Major browsers in the time of writing of this thesis are according to the [4]: Google Chrome, Mozilla Firefox, Internet Explorer, Safari, Opera
4. Application containers are special programs dedicated to provide a runtime environment for complex enterprise web applications, e.g. JBoss AS, Wildfly, Apache Tomcat

2.2 Need for automation

comparison of hiring people to do manual testing vs. automated testing cost

3 Analysis of existing solutions

How the process of testing with these tools looks like, its advantages and disadvantages.

3.1 Mogo

3.2 BBC Wraith

3.3 PhantomCSS

3.4 Facebook Huxley

3.5 Rusheye

3.6 Drawbacks

Conclusion of drawbacks, and why we try to propose another approach

4 New approach

4.1 Hypothesis

Simply: reuse of functional tests of the application for visual testing

4.2 Process

How one would use my tool and where in testing stack such visual testing has its place, written in business process notation

4.3 Analysis of useful tool output

Requirements for useful output of such a tool based on questionnaire for RichFaces team, or maybe I will ask all JBoss employees

5 Implemented tool

An answer to the new process, requirements: CI viable, reusing what can be reused, extensible, cloud ready, multiple users

5.1 Client part

5.1.1 Arquillian

Integration testing, starting containers, event based machine

5.1.2 Arquillian Graphene

Functional testing of Web UI, screenshooter

5.1.3 Rusheye

Screenshots comparison, rewritten to Arquillian core

5.1.4 Graphene visual testing

An adaptor between Rusheye and Arquillian Graphene

5.2 Server part

5.2.1 Web application to view results

Its architecture, reasoning for chosen solutions, screenshots of app, key functionality

5.2.2 Storage of patterns

Description of solution, reasoning

6 Deployment of tool and process

6.1 Deployment on production application

Deployment on stable app

6.2 Deployment on development application

Deployment sooner on application which is in Alpha phase, my hypothesis is that it will not be worth to deploy it on such a app, due to too many changes

6.3 Usage with CI

Jenkins job and its cooperation with the tool, more particullary tool ability to handle multiple jobs, apps, versions, etc.

6.4 Cloud ready

The app can be easily deployed on Openshift

6.5 Results

The percentage of improvement of QA effectiveness

7 Conclusion

What I developed, What I improved, What can be better, Possible ways of extensions: Openshift cartridge

Bibliography