Expertise
and insight
for the future

Jose Cezar Ynion

Using AI in Automated UI Localization Testing of a Mobile App

Metropolia University of Applied Sciences

Master of Engineering

Information Technology

Master's Thesis

07 April 2020

PREFACE

As a Software Engineer, I always get curious about the latest trends in the IT field, may it be about the new programming language, paradigms, or tools. And nowadays, AI is a hot topic with so much hype around it. So I asked myself, "Beyond Boston housing data and MNIST exercises that I did, how can AI help me in my daily work?".

Over the years, my work tasks are mostly related to UI, from Visual Basic to Visual C++, QT, HTML5, now Swift, and I am about to start learning SwiftUI. UI automated testing already brings a lot of challenges, much more on the localization. If AI is already solving complex problems today, for sure it can improve my productivity, how about automating the localization testing?

The research topic blends my interest and work experience. This will not be possible without the support of my manager and colleagues at work.

To my wife, my son Gio Raphael and soon to be born daughter, *thank you for allowing me to spend time writing this thesis. Those were precious times that I should have spent with you*.

Dedicated to my mother and father, my first teachers. *Nanay and Tatay, we have come a long way*.

Espoo, 07.04.2020
Jose Cezar S. Ynion

Author Title Number of Pages Date	Jose Cezar Ynion Using AI in Automated UI Localization Testing of a Mobile App 53 pages 7 April 2020
Degree	Master of Engineering
Degree Programme	Information Technology
Instructor(s)	Antti Koivumäki, Senior Lecturer

Localization testing is seldom addressed in the scientific literature, especially on the mobile app domain. This thesis focused on the practical implementation of an automated localization testing system for an iOS mobile app.

I work as a Software Engineer in an international company that has mobile and desktop apps as main products. Each app is localized into multiple languages. Testing that each User Interface (UI) displays the right content per language is the most time-consuming part of the software development lifecycle. Due to the visual nature of the tests, this is done manually and repeatedly in different devices with various Operating Systems and screen resolutions.

Effectively testing the localized app is always a challenge for Quality Engineers because they are not language experts. The scope of the tests is somewhat limited to finding bugs like wrong layout, overlapping, untranslated texts, and wrongly represented characters.

The prototype system called NEAR is the outcome of this thesis. It was designed to automate most of the tasks in testing UI Localization. It integrates pre-trained cloud-based Artificial Intelligence models of Natural Language Processing (NLP) and Computer Vision from service providers like Google to add visual context to a test. As a result, the time required to run the regression test is less. The scope of the testing now includes finding bugs that need linguistic skills like mistranslation, text truncations, and locale violations.

Keywords	A.I., automated testing, localization testing, i18n, l10n, mobile app testing, computer vision, natural language processing

Contents

List of Abbreviations

AI	Artificial Intelligence
CI	Continuous Integration
CV	Computer Vision
DL	Deep Learning
DOM	Document Object Model
I18n	Internationalization
L10n	Localization
MBT	Model-Based Testing
ML	Machine Learning
NLP	Natural Language Processing
OCR	Optical Character Recognition
OS	Operating System like Android, iOS, Windows, Linux and OSX
SDK	Software Development Kit
TA	Test Automation
UI	User Interface
UX	User Experience
QA	Quality Assurance
QE	Quality Engineers
YOLO	You Only Look Once

1 Introduction

1.1 Background

Every mobile app developer wants to make their app a global success. The first step to gain a wider audience is by making the app available in all App stores' supported countries or regions. However, top online store locations that have a significant market share aside from the US are non-native English speakers. With millions of apps to choose from, one common strategy that software companies use to stand out is to make their product display the content according to the local market.

This process of adapting an app's User Interface (UI) to a different language and region so that it can be understood on a local market is called localization. It typically involves translating all the texts, replacing icons and images, and presenting the correct date/time format in the target language and culture. UI is an integral part of the User Experience (UX). UX is defined as "user's perceptions and responses that result from the use and/or anticipated use of a system, product or service" (ISO9241-210, 2019). UX can affect the app's success. It determines whether the potential customer will use or discard the app (Applitools, 2019).

Localization process usually starts with exporting the resource like texts, then sending those to an external Localization service provider. Localized resources are then imported back, and the app is built and tested afterward. Exporting and importing the resources are typically automated. This process is part of the Continuous Integration (CI) of the app development cycle. However, there is still a lack of automation in testing.

Developers make the app ready for localization, then Quality Engineers (QE) validate it for correctness. However, QEs are not language experts. They can detect bugs such as untranslated texts, texts that are overlapping due to lack of space, and misaligned UI elements. However, bugs that are hard to detect, like mistranslation or wrong context, truncation, wrong date/ time format, requires linguistic expertise. (Carmi, 2019)

Quality Engineers repeat this time-consuming task for all supported languages. They are doing the same tests on different devices with various screen sizes, resolutions, and also with several OS versions. QEs resort to semi-manual testing due to the visual nature of

the tests, where most tools cannot adequately find bugs. This research is about finding ways to minimize the manual work, speed up the testing time, and effectively find bugs in UI Localization testing.

1.2 Motivation and Research Problem

In a nutshell, every element in the User Interface can be categorized into groups such as buttons, labels, and icons. Various platform-specific tools already exist to extract data from these elements. Finding ways to analyze this localized data for correctness without human interaction is a challenge. Artificial Intelligence (AI) is one field that might solve this challenge because data is where the AI shines.

Recent advancement of AI in the fields of Computer Vision, Natural Language Processing (NLP), is auspicious. One of its practical usages might be to augment the linguistic and visual skills required to do localization testing. Big companies like Google, Microsoft, and Amazon are investing heavily to forward these fields of sciences and already provide platforms for developers to integrate it with their products (Kukushkina, 2019).

This thesis have two primary goals. The first goal is to expand the scope of localization testing that Quality Engineers are manually doing in the case company. The second goal is to reduce the time allotted for testing.

It was not known during the writing of this thesis, whether AI could help to expand the scope of the test. Moreover, the question is if AI has subfields that can provide the visual contexts to the test and add linguistic skills needed.

It is proven that test automation can reduce the time allocated for testing, along with other benefits. However, localization testing is rarely addressed in the scientific literature, thus finding existing tools, especially for the mobile domain, is a challenge. Likewise, there is a risk that none of the manual testing tasks can be automated.

Those two goals formed the following research question:

Can AI be used to improve UI Localization testing?

The outcome of this research is a proof-of-concept automated system to test UI Localization of a mobile app. The scope of this research is to investigate the existing UI automation tools, evaluate and integrate the pre-trained AI models from service providers like Google and Microsoft, and test the system with at least 2 example iOS apps that are localized in 1 language aside from English.

This research aims to find ways to improve the process of UI Localization testing by a) cutting the amount of time it requires from QE to test, b) the possibility to achieve 60% automation, and c) expand the testing scope.

1.3 Research Process, Method and Material

This research follows the Design Science approach to create an innovative solution. This subchapter describes the research process from data gathering up to the evaluation of the proposed solution.

1.3.1 Research Process

This research was conducted in stages (Figure 1). The first stage was finding the current state by gathering metrics like time spent, average bugs found, and types of issues. Likewise, a preliminary study was done to familiarize with the tools used in UI testing. Requirements were narrowed down, and at the same time, the areas for improvement in the existing process were identified. Information and data came from the interviews with Quality Engineers and internal documentation from the case company.

The second stage was getting acquainted with the theoretical background. Previously published research about localization testing, automation, and UI testing were examined, and compared with each other. Next was evaluating the suitable pre-trained models of NLP and Computer Vision from AI platform service providers like Google, Microsoft, IBM, and Amazon.

The third stage was building a solution. Trial and error were conducted with the chosen tools and technologies, then selecting the best ones that were easy to integrate, covered the test cases that QEs require, and faster to execute.

In the last stage, the prototype system was demoed, and results were evaluated.

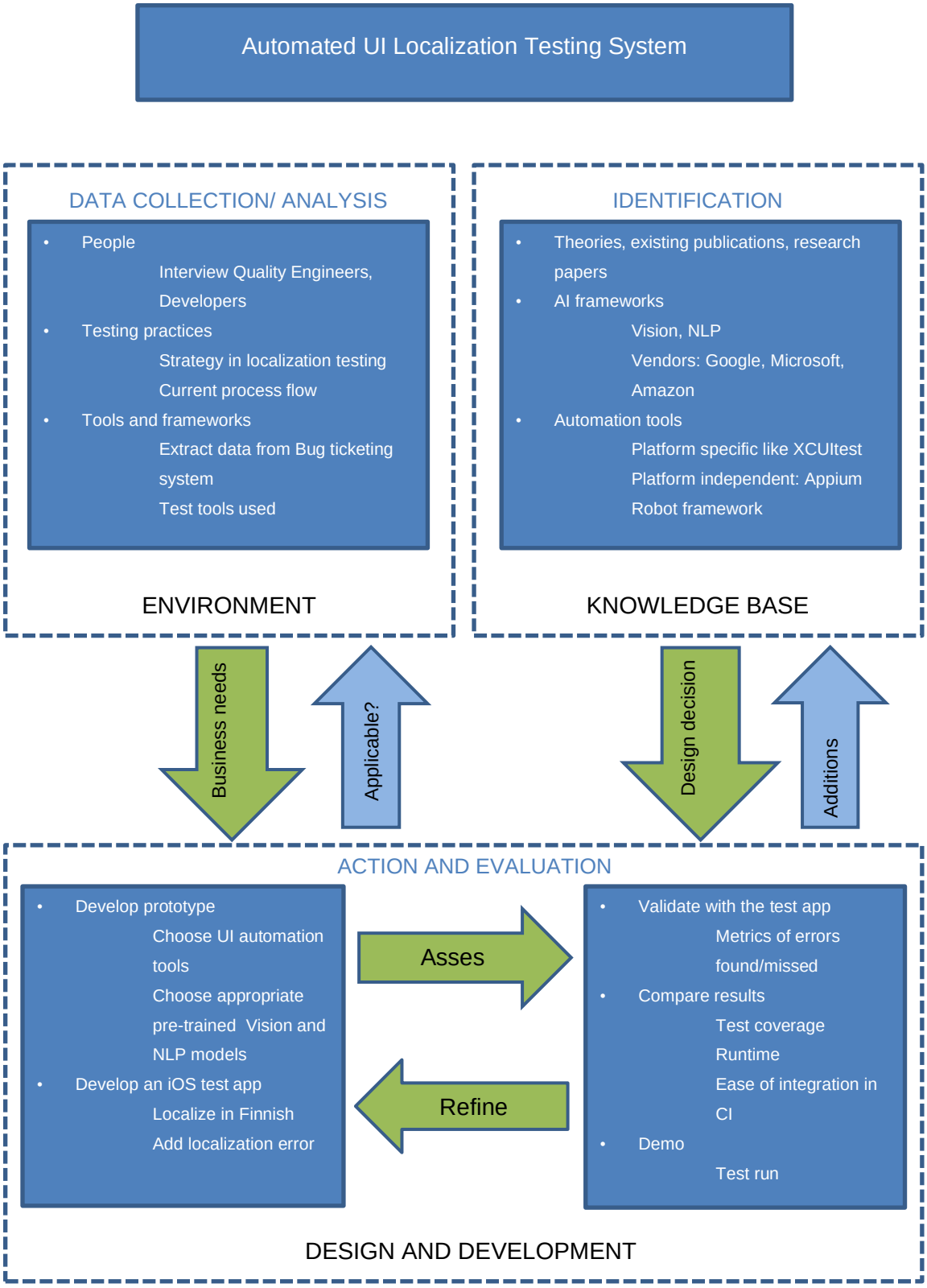


Figure 1. Research plan adapted from Henver et al. (Hevner, et al., 2001)

1.3.2 Method and Material

A qualitative method was used to gather the information that defines the scope of the research question. Requirements for the solution came from observations from the current test process and system, interviews with the Quality Engineers, and project lead. Internal documents and bug tickets were examined to know the types of bugs found and missed during testing, and likewise know the corresponding severity for prioritization.

The solution was evaluated using the quantitative method. The test runs determined the test time reduction, the number of bugs found, bugs missed, and the number of false-positives.

1.4 Organization of the Thesis

This thesis is divided into five chapters. Chapter 1 introduces the goals, motivation and expected outcome of this research. Chapter 2 presents the theoretical background of the existing work done in a field of UI Localization testing, AI fields of Computer Vision, NLP, and major service providers for these technologies. Chapter 3 discusses the requirements and steps in building the proof-of-concept UI Localization testing system. Chapter 4 then discusses the validation of the results. Finally, the last chapter is the summary and the conclusion of the thesis.

2 Theoretical Background

This chapter presents the core concept of the localization process. It explores the various industry-standard practices and strategies in testing localized software, including previous research that focuses on automating it. Moreover, to come up with an answer to the research question, information is analyzed and compared from publications, journals, websites, and books about Automated UI Testing and AI usage in UI Automated Testing. Specific subfields of AI, such as Computer Vision and NLP, are also discussed. The knowledge presented here lays the ground for understanding, scoping, and designing the Automated UI Localization Testing System that this thesis aims to implement.

2.1 App Internationalization and Localization

There are two main steps to design an app for a global audience. The first step is to *internationalize* the app, and the second is to *localize* it. Internationalization and Localization are sometimes written as *i18n* and *l10n*, respectively, where 18 and 10 are the number of letters between the first and last character of each word (w3c, 2005). Internationalization and localization are sometimes referred to as *globalization* (Hardy, et al., 2012).

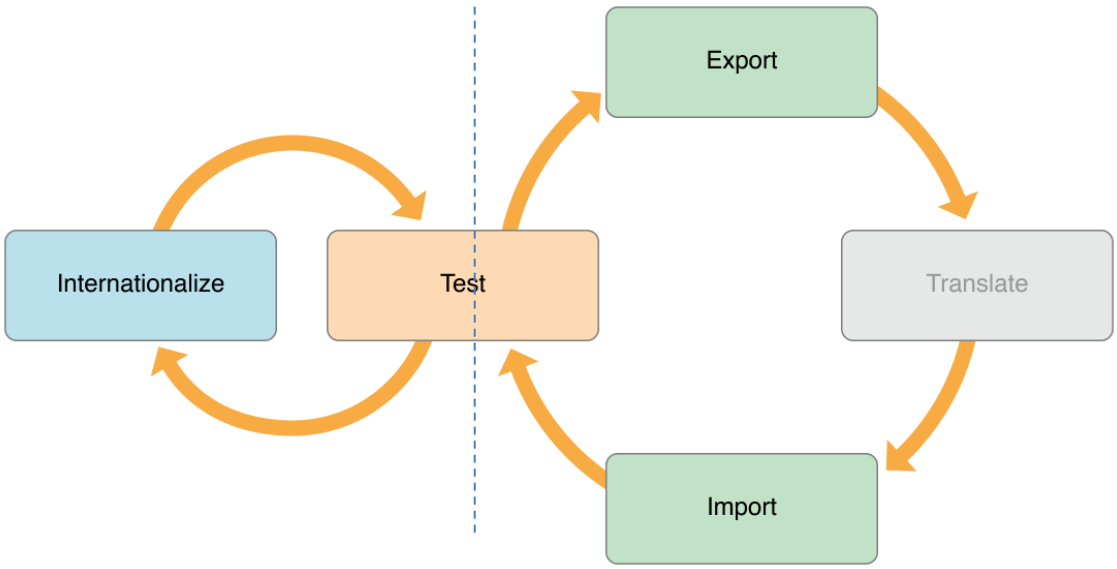


Figure 2. Localization process (Apple, 2015)

2.1.1 Internationalization Process

Internationalization is the process of preparing the app to adapt to different languages, regions, and cultures (Apple, 2015). It means that it should be able to display text, numbers, and currency in appropriate locales. A locale is a combination of language and region (Android, 2019). It represents cultural conventions (Flanagan, 2002).

Internationalization is a pre-requisite for localization. The following are the typical activities involved in internationalization.

Auto Layout. Adjusting or resizing view layouts to accommodate longer strings. UI components that display text must not have a fix width or height. Some languages have a longer localized text, and this may be truncated if the control's width or height is not flexible.

Externalize Resources. Putting the user-facing content into resource files. Separating the localizable element from the code, such as text, images, and videos.

```
/*
Localizable.strings (Finnish)
*/
"hello_world" = "Hyvää huomenta!";

/*
Localizable.strings (English)
*/
"hello_world" = "Good morning!";


let greetingsId = "hello_world"
let goodMorning = NSLocalizedString(greetingsId, comment: "")

print(goodMorning)
```

Listing 1. Load and print the string according to the system's language. This will print 'Good morning!' if the system's language is English. Otherwise, it will print 'Hyvää huomenta!' if it is in Finnish.

System-Provided Formatting Methods. Changing the code to adhere to locale formats when displaying data such as date, time, numbers, personal names, and forms of address. Confusion will arise if this is not done. Awwad and Slany's mentioned in their

research that “typical problems are ‘../../....’ date formats between the US and European date formats, where it is unclear whether 10/2/2016 is the 2nd of October (US) or the 10th of February (most European countries) 2016” (Awwad & Slany, 2016).

```
let epoch: TimeInterval = 1582890404
let date = Date(timeIntervalSince1970: epoch)

let formatter = DateFormatter()
formatter.dateStyle = .short
formatter.timeStyle = .short
formatter.timeZone = TimeZone.ReferenceType.default

print(formatter.string(from: date))
```

Listing 2. Format a date value according to the system's region. This will print '2/28/20, 1:46' PM if the region is the US, and ‘28.2.2020 13.46’, if the system's region is Finland.

User Interface Mirroring. For right-to-left languages, mirror the user interface and change the text direction as right-aligned. The reading order for the speakers of bi-directional languages is from right to left.

Figure 3. Example of right to left UI (Awwad & Slany, 2016)

However, according to Apple Developer Guide, some elements must not flip, these are:

- “Video controls and timeline indicators
- Images, unless they communicate a sense of direction, such as arrows
- Clocks
- Music notes and sheet music
- Graphs (x– and y–axes always appear in the same orientation)”. (Apple, 2015)

2.1.2 Localization Process

Localization is a process of translating an app into different languages. Resources such as text, audio, and images are exported and then submitted to translators. When translations are ready, they are then imported back to the app. Exporting and importing varies depending on the platform. It can be as simple as copying the files or using platform-specific developer tools like XCode, as illustrated in the figure below.

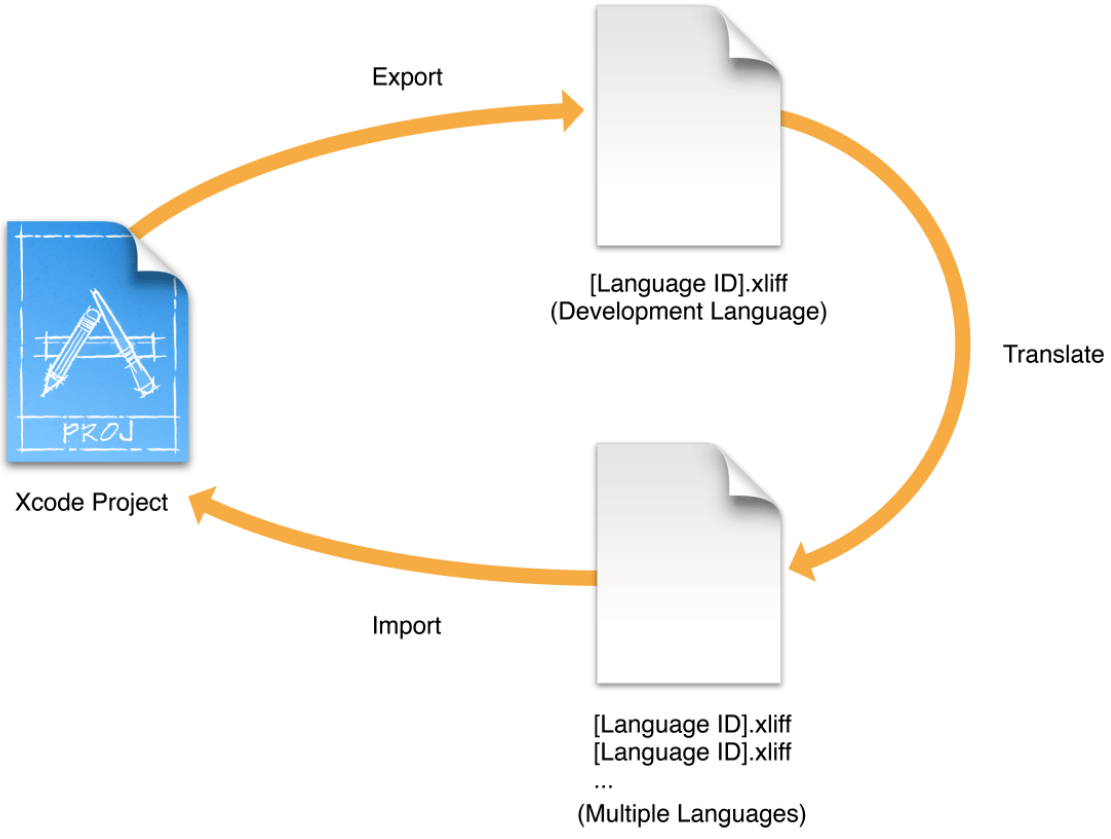


Figure 4. Exporting and Importing of string resources for iOS or OS X app. (Apple, 2015)

Translation step is commonly outsourced to a third-party localization service provider, or inhouse if there are language experts. Google even integrated an App Translation

Service in its Google Play Console. According to the manager that was interviewed during this research, the translation process includes a validation round inside the vendor that provides translation. If screenshots are available, review rounds are used to validate further the string in projects. He also pointed out the common issues during translation step such as:

- Difficulties in translation due to poor internationalization of the product or the resource file.
- Unclear or poor English combined with sentences that were split.
- Unclear variables and configuration information in the resource file.
- Inflexible UI layout design causes the majority of issues.
- Typos and mistranslations due to lack of context.
- Highly specialized and new terminology.

2.2 Localization Testing

A wrongly worded or grammatically wrong text can ruin the User Experience of an app despite its sophisticated features. The quality of the app depends on the localization level, and it cannot be stressed enough the importance of localization testing in quality assurance of a localized app (Zhao, et al., 2010).

2.2.1 Testing Strategies

Test strategies can vary at each stage of the globalization process. Nevertheless, the pre-requisite is the test environment. It must be properly set up to uncover issues specific to culture, language, date and time format, and bi-directional language. Test environment can be either an Emulator or a physical device. Emulators simulate a mobile device on a laptop or PC (Haller, 2013). A test device's locale must be set to the target language and region to test a localized app. Android Developer Guide (Android, 2019) also suggests creating a custom locale that is not supported by the system to test how the app runs. It must display the default resource.

Pseudo-localization is a common technique to test the app during the internationalization stage of app development. “The pseudo-localization process replaces the characters in a given source string (such as in an English language string) with characters from a target set (such as Unicode) and changes the size of the string by adding extra

characters to it" (Gundepuneni, et al., 2012). This method reveals whether elements in the UI can resize properly with string length variations, and adapt to different language fonts. If the UI displays un-pseudolocalized text, then it means that there are untranslatable messages in your source code (Android Developer Guide, 2019). It is a technique to test the readiness of the app while waiting for the localization.

The following are common issues with a localized app:

- Non-localized strings. Hardcoded strings are not sent to translation.
- Long texts that can break the UI layout. Label or text elements might overlap.
- Wrong person's title or postal address format.
- Wrong currency, number, date or time format.
- Right-to-left layout if elements are not mirrored.

Android Developer Guide summarizes the best practices to test the app.

- "Where possible, always use native-language speakers to test your localization.
- On each test device, set the language or locale in Settings. Install and launch the app and then navigate through all of the UI flows, dialogs, and user interactions. Enter text in inputs.
- Look out for clipped text or text that overlaps the edge of UI elements or the screen.
- Verify that text is line wrapped appropriately.
- Check for incorrect word breaks or punctuation.
- Validate alphabetical sorting to ensure the order is as expected.
- Make sure all layouts and text directions are correct.
- Watch for untranslated text; check that the resources directory is marked with the correct language qualifier.
- Test for default resources." (Android Developer Guide, 2019)

2.2.2 Previous Work on Automated Localization Testing

Searching for keywords such as "*localization*", "*localisation*", "*globalization*" together with "*automated testing*" from Metropolia's digital libraries and resources yields a minimal result. Ramler and Hoschek also pointed out that there is very little scientific literature focusing on localization (Ramler & Hoschek, 2017). However, localization testing is the

candidate for automation because it involves many repetitive tasks. As an example, Archana et al. (Archana, et al., 2013) enumerated the following issues that the automation system can detect from a web-based app:

Inconsistent font usage. Small font can result in unreadable text or text that can appear garbled.

Character corruption. Presence of mojibake (garbage characters), tofu (hollow boxes) due to wrong encoding or missing glyph for that character from the chosen font, respectively.

Figure 5. Character corruption (Archana, et al., 2013)

Hardcoded texts. Texts that are not translated according to the locale.

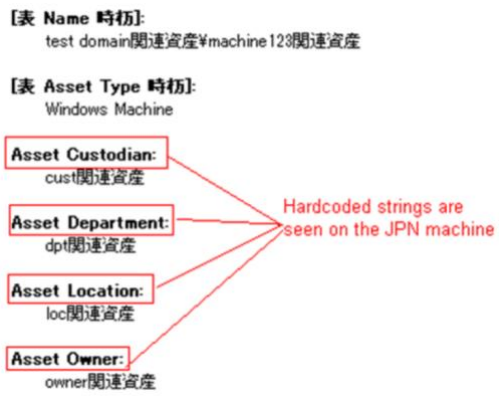


Figure 6. Hardcoded strings (Archana, et al., 2013)

Over translations. Strings that should not be translated are not presented according to the value from the app resource. These are default strings like product name and versions.

Automated localization testing can find not only cosmetic issues but also critical bugs. A simple truncation issue can lead to a misleading situation. As an example "110V" voltage value is shown as "10 V" in a right aligned text where there is not enough space to accommodate the number value. (Ramler & Hoschek, 2017)

GWALI (Global Web Applications' Layout Inspector) has likewise proven that a presentation failure of web apps can be detected by automation. GWALI is a prototype for detecting distortion in a web page's appearance caused by internationalization. It can narrow down the HTML elements or text that is causing the problem. This tool identified 91% of defects based on their test results and has a running time of 9.75 seconds per web page. Their approach was to build Layout Graphs and comparing these graphs to identify the distorted appearance of a webpage after localization. (Alameer, et al., 2016)

Figure 7. Part of a webpage and its localized version (Alameer, et al., 2016)

Figure 8. Text overlapping with a button after translation (Alameer, et al., 2016)

There are not that many research papers related to automated localization testing for mobile apps, especially for iOS.

2.3 Automated UI Testing

The artifact of this research is a prototype of an automated localization testing system. The test cases for this system are variants of UI tests because localized resources, such as strings and images, are presented in the UI through elements like buttons, text labels,

and icons. Therefore, verifying that localized data are displayed correctly is considered as a UI testing task.

It is essential to confirm that the UI meets functional requirements and consistent in style. However, manually testing it is time-consuming, tedious, and error-prone. Automating user action is an efficient way to test the app (Android, 2019).

An automated UI test case is a coded test that generates user actions or events such as typing in a text field, swiping views, and tapping buttons. It then validates the changes in the user interface or functionality of the app according to the expected outcome of the action. Automated tests are fast and repeatable. Aside from testing the app flow, automated UI tests can check visual consistencies of UI elements properties; this includes but not limited to: colors, icons, fonts types, and font sizes.

According to Microsoft, “Automated tests that drive your application through its user interface (UI) are known as coded UI tests (CUITs)” (Microsoft, 2016). The app is first tested manually, and then this scenario will be automated. The figure below illustrates the different use cases of coded tests depending on the functionality being tested.

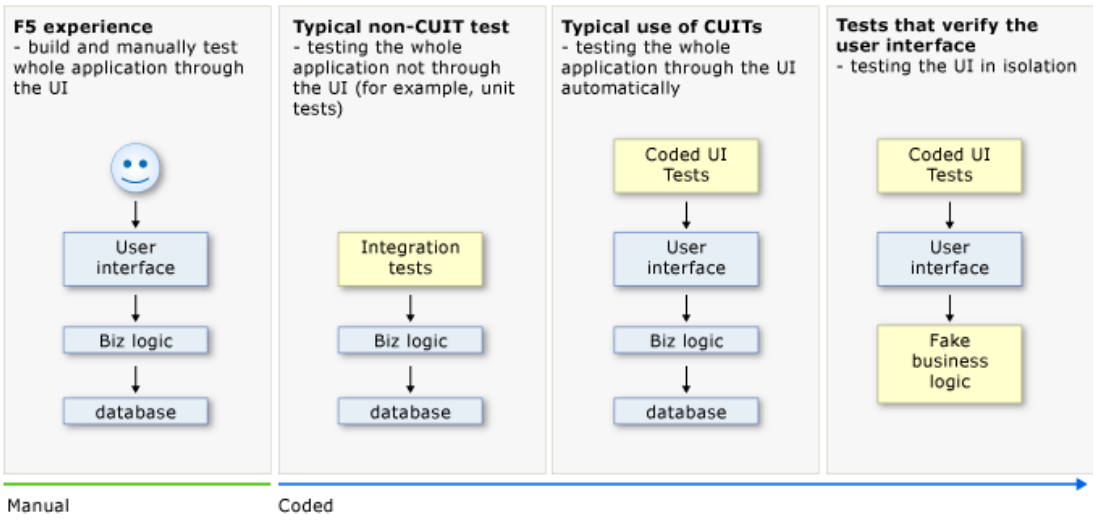


Figure 9. Typical flow and approaches of test development. (Microsoft, 2016)

2.3.1 Test Case Generation

Quality Engineers or developers create a set of test cases to exercise the functionality of the app. Test Automation Framework will automate the execution of these tests. One of the challenges is to achieve high coverage due to a large number of execution paths,

which means engineers need to create a lot of test cases. Most UI test tools provide a “record” feature that allows humans to manually explore the app and later generates a code to “replay” their actions.

A test automation framework is a set of concept and tools to create tests and perform automated software testing (Archana, et al., 2013). UI Test Automation Frameworks address two requirements for automated testing (Microsoft, 2019):

1. *Locate a specific view*. This is performed through queries. A test case must be able to query for a view or element from the screen. The framework should be able to return this view object so that actions can be done to it.
2. *Interact with a view*. APIs to perform actions on a view such as tapping, entering text, or swiping.

Google and Apple provide UI test frameworks tailored for their respective platform, XUITest for iOS and Espresso for Android. Test cases created for these frameworks must be written in a programming language specific to their particular platform. Example codes are listed below.

```
func testRefresh() {
    let app =  XCUIApplication()
    app.launch()

    XCTContext.runActivity(named: "Select Refresh") { _ in
        app.navigationBars.buttons["refresh"].tap()
    }

    XCTContext.runActivity(named: "Check the # of items") { _ in
        let cells = app.tables.cells
        XCTAssertEqual(cells.count, 4)
    }
}
```

Listing 3. Test case for iOS app written in Swift.

```
public void testEspresso() {
 // Check if view with the text 'Hello.' is shown
 onView(withText("Hello.")).check(matches(isDisplayed()));
 // R class ID identifier for 'Sign in' - and click it
 onView(withId(getInstrumentation().getTargetContext().getResources()
 .getIdentifier("com.twitter.android:id/sign_in", null, null))).perform(click());
```

Listing 4. Espresso sample test code snippet. (Bezmolna, Victoria, 2019)

These frameworks are stable and no cost for setup as it is usually bundled together with the app development tools. Cross-platform frameworks also exist like Appium that can run a test script for either Android or iOS. Its advantage is that it supports many programming languages to write a test case and run parallel Android UI tests (Bezmolna, Victoria, 2019). However, it is complex to setup initially, test runs are slow and can have compatibility issues with every update of platform tools like XCode (Mischinger, Sarah, 2019).

```
def test_should_send_keys_to_inputs(self, driver):
    text_field_el = driver.find_element_by_id('TextField1')
    assert text_field_el.get_attribute('value') is None
    text_field_el.send_keys('Hello World!')
    assert 'Hello World!' == text_field_el.get_attribute('value')
```

Listing 5. Appium sample test case written in python. (Appium, 2019)

Another approach to automated testing is Model-based Testing (MBT). This approach is about automating test case generation. It requires a model as input in order to generate test cases. Morgado and Paiva highlights two main issues of MBT, those are: “1) the necessity of an input model from which test cases are generated and whose manual construction is a time consuming and error prone process and 2) the combinatorial explosion of generated test cases” (Morgado & Paiva, 2015). Their paper presents a solution through reverse engineering implementation. It identifies the UI Patterns and based on that, applies a similar test strategy from their catalog of patterns, and continue exploring the app. Another issue pointed by Arnatovich et al., is that the existing model-based testing tools for Android generates trivial or non-sensible input, or sometimes it requires the user to provide such data during app testing (Arnatovich, et al., 2016).

2.3.2 Test Case Execution

In the Continuous Integration (CI) process, developers commit their code changes to a central source repository several times per day. This event triggers an action to run the automated tests. CI runs the automated test to verify that new code changes do not break existing features or introduce new bugs, so the software remains deployment ready at all times (Atlassian, 2020).

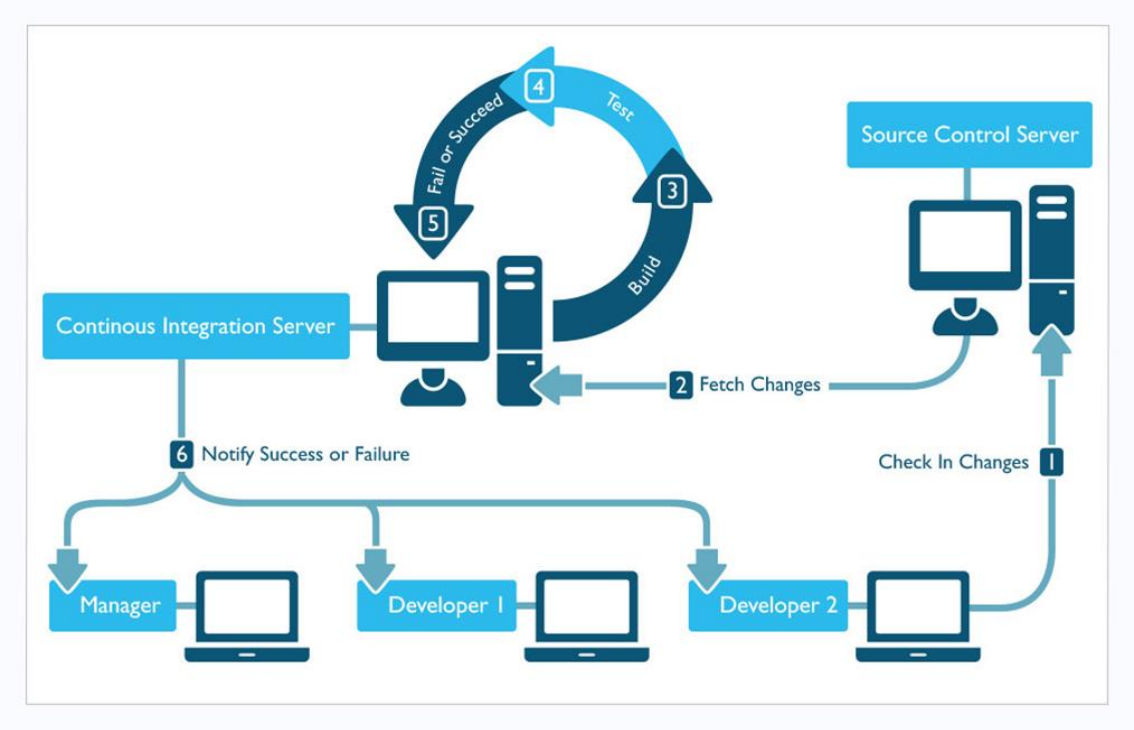


Figure 10. Technical implementation of Continuous Integration (pepgotesting, ei pvm)

UI Tests can be executed either in real devices or in virtual devices such as simulators and emulators. An emulator is a virtual representation of the entire device, including the low-level system calls. On the other hand, a simulator runs a version of mobile OS implementation in the host machine's kernel. These virtual devices are software programs, and tests that are performed on it will not uncover device-specific bugs. Customers' use cases can only be performed on a real device such as network change events, phone calls, push notifications, audio input/ output, and among others. However, procuring real device is expensive, and needs to be updated as new devices come to the market very often. (SauceLabs, 2018)

These environments may either reside on-premise or on the cloud. Cloud-based test labs enable customers to use a set of devices based on the subscription plan. The advantage of using service from cloud is that there is no need to maintain and purchase the latest devices. (Garg, 2016)

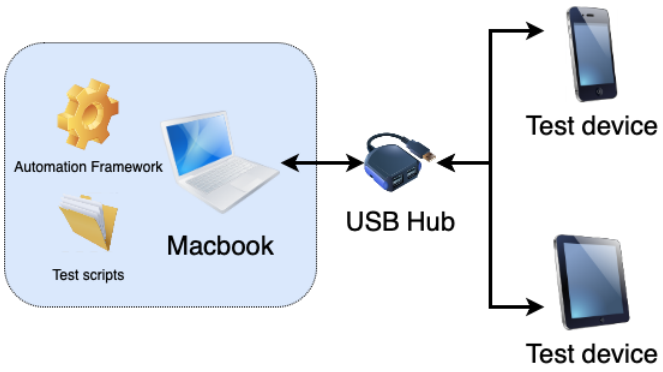


Figure 11. Physical setup of the test environment on-premise that can support iOS and Android app testing

Cloud-based test infrastructure allows running of tests in parallel against a massive collection of physical devices. Facebook (Facebook, 2016) mentioned that they require 2000 mobile devices to cover all combinations of device hardware, operating systems and network connections. Cloud Testing Service providers also provide better analytics and reporting features, they also solve complicated signing issues with Apple's app security model, and the infrastructure is scalable (SauceLabs, 2018). AWS Device Farms (Amazon, 2020) even allow debugging to reproduce issues and can interact with a device via a web browser.

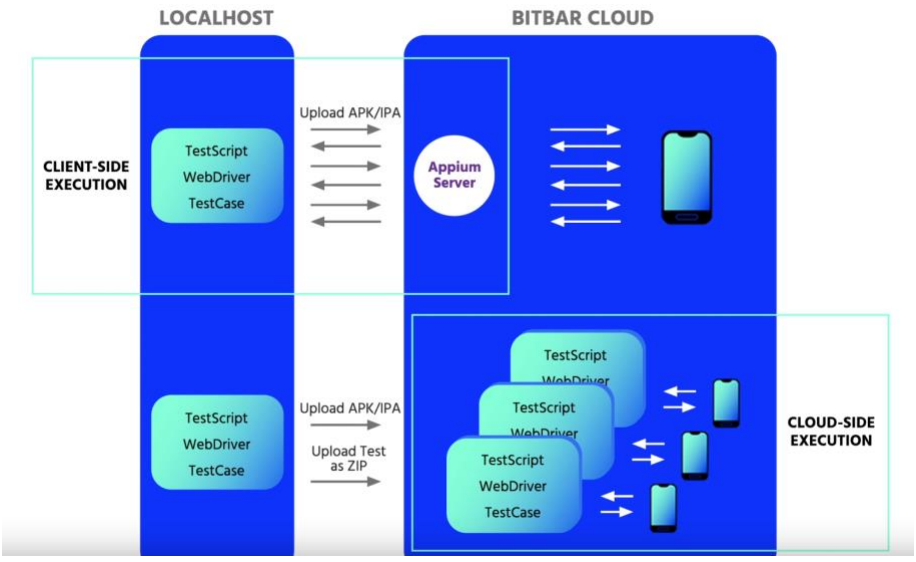


Figure 12. Bitbar cloud-based testing infrastructure (Bitbar, 2019)

2.4 AI in Automated Testing

To answer the research question of whether AI can be used to improve UI localization testing, it is fundamental to understand what AI is and how it is providing useful solutions to software test automation domain.

Testing approaches evolved over the years, from testers acting as product users, interacting with the application to coding test scripts for automation. In the future, test automation techniques would involve predictive analysis, self-remediation, cognitive automation, and machine learning according to 38% - 42% of the organizations surveyed in World Quality Report 2017 (Sogeti, Capgemini, Micro Focus, 2017).

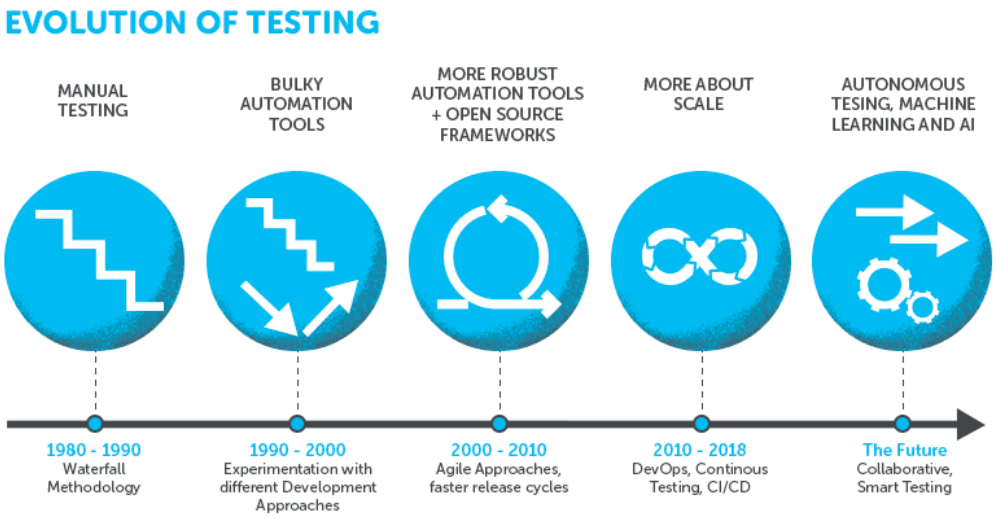


Figure 13. How testing has evolved over the last 4 decades. (Testim, 2018)

World Quality Report 2019-2020 (Capgemini, Sogeti, 2019) recommends building a smart, connected test platform with intelligent analytics. According to the same report, Artificial Intelligence (AI) can make testing smarter. However, the test team needs AI-related skillset, like data science, statistics, and mathematics. Integrating AI in Software testing is a natural progression (Testim, 2018).

2.4.1 AI Basic Concepts

Researchers have no exact definition of AI. However, a system with AI has two key attributes. First is *autonomy*, which means it must be able to perform tasks in complex environments without constant guidance from the user. Second is *adaptability* or the ability to improve by learning from experience. (Reaktor, University of Helsinki, 2018)

Although AI covers various theories and technologies, the two main classifications are Machine learning and Deep learning (Taulli, 2019). Machine learning (ML) is a subfield of AI and defined as “Systems that improve their performance in a given task with more and more experience or data” (Reaktor, University of Helsinki, 2018). ML deals with constructing a system where the focus is on learning from available data or reactions of the environment. One of the subfields of ML is Deep learning (DL). It “refers to certain kinds of machine learning techniques where several "layers" of simple processing units are connected in a network so that the input to the system is passed through each one of them in turn” (Reaktor, University of Helsinki, 2018).

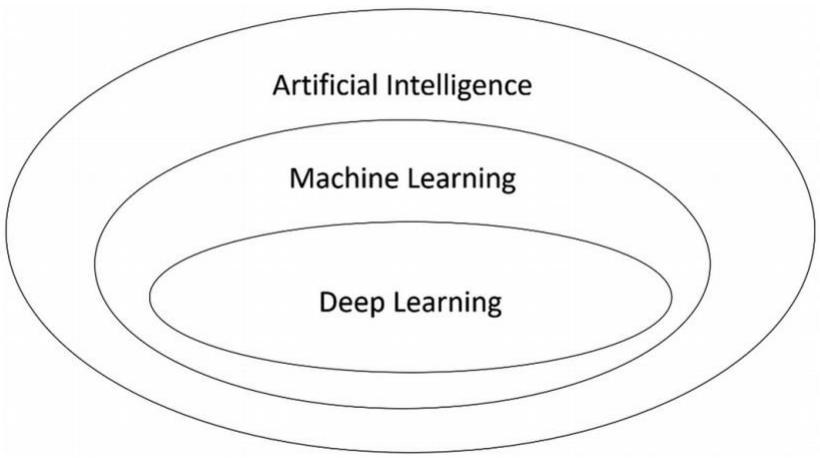


Figure 14. “High-level look at the main components of the AI world”. (Taulli, 2019)

The book *Artificial Intelligence Basic: A Non-Technical Introduction* (Taulli, 2019) illustrates better the distinction between deep learning and machine learning through its example of finding a picture of a horse from thousands of animal pictures. In machine learning, the model must be trained by using labeled photos of animals as its training data. It can also employ feature extraction, a process of analyzing the pixel patterns of the images itself to develop the characteristics of a horse. On the other hand, the Deep

learning approach analyzes all the data to find the relationships between pixels. It will use a neural network, just like the human brain. (Taulli, 2019)

Machine learning is already applied in many applications. Some examples (illustrated below) are *Predictive maintenance* - to forecast when equipment will fail. *Customer experience* - to leverage the data to gain customer insights on what really works. *Finance* - to detect discrepancies in billing. (Taulli, 2019)

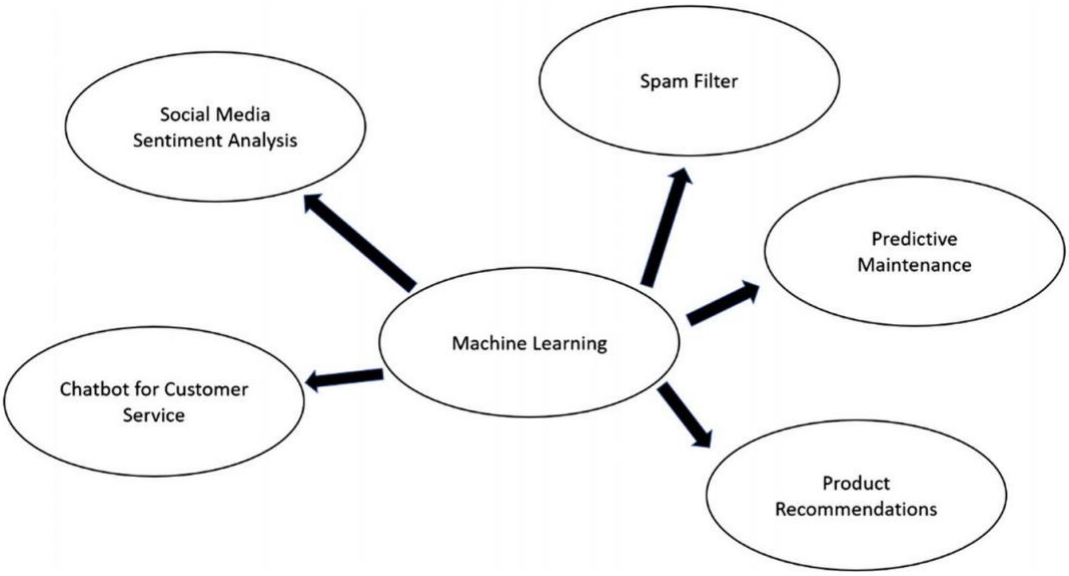


Figure 15. Applications for machine learning. (Taulli, 2019)

The AI infrastructure requires a lot of computing power to train the neural network. It also requires a lot of data for algorithm to perform better. These two are the major stumbling blocks for smaller companies to build, implement, or adopt AI in their business. Tech giants companies like Amazon, Google, IBM, and Microsoft are addressing these challenges by providing cloud-based AI services. IBM referred to this as a “cognitive-as-a-service”, and according to their study, this setup is the preferred way by most early adopters in developing and delivering AI-infused solutions (IBM, 2016).

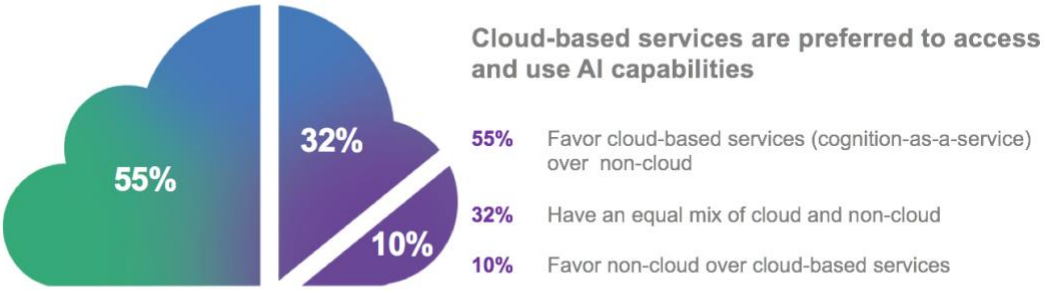


Figure 16. Preferred way to access and use AI capabilities (IBM, 2016)

Aside from the lower cost compared to building non-cloud infrastructure, cloud-based AI service also has other benefits. Risk reduction is one of the benefits, which means if the product is not successful, a company can terminate the service without worrying about the expensive hardware equipment or the data scientists that they do not need anymore. Similarly, if the product is a success, a company can expand or scale their infrastructure on demand. Another advantage is access to bleeding-edge technologies. Major cloud vendors have a large scale investment in research and development. Technologies can become obsolete fast, and these major vendors can roll out new capabilities regularly. (V2Soft, 2018)

As mentioned in the introduction section, this thesis aims to use multiple cloud-based AI service. Listed below are services that are considered relevant to this research.

- **Google Cloud Vision.** It offers pre-trained machine learning models through REST APIs. It can classify and assign labels to images. Likewise, it detects objects and faces and reads printed and handwritten text from images.
- **Google Natural Language Processing.** It "uses machine learning to reveal the structure and meaning of the text" (Google, ei pvm). The pre-trained models can extract information, understand sentiments, and parse intent from customer conversations.
- **Google Translation.** "Dynamically translate between languages using Google's pre-trained or custom machine learning models" (Google, ei pvm).
- **Amazon Rekognition.** It uses deep learning technology. It "can identify objects, people, text, scenes, and activities in images and videos, as well as detect any

inappropriate content" (Amazon, 2020). It also provides facial analysis and facial search capabilities.

- **Amazon Textract.** It is a document text detection and analysis service using deep-learning technology. Its API can "detect text in a variety of documents, including financial reports, medical records, and tax forms" (Amazon, 2020). It can also extract forms and tables for documents with structured data.
- **IBM Watson Natural Language Processing**. It uses deep learning-based NLP models like named entity recognition, sentiment analysis, keyword extraction, part-of-speech tagging, topic modeling to analyze text to extract metadata from the content.
- **Microsoft Azure Cognitive Services.** A comprehensive portfolio of domain-specific AI capabilities with a set of APIs for vision, language, speech and search capabilities.

2.4.2 Computer Vision in Automated Testing

"Computer vision allows machines to identify people, places, and things in images with accuracy at or above human levels with much greater speed and efficiency. Often built with deep learning models, it automates extraction, analysis, classification and understanding of useful information from a single image or a sequence of images. The image data can take many forms, such as single images, video sequences, views from multiple cameras, or three-dimensional data" (Amazon, 2020). Computer vision (CV) typically imitate the visual perception of humans, intending to interpret natural scenes of images (Peters, 2017).

The computer vision process typically starts with acquiring a large set of images from real-time video or photos. It then uses deep learning to process the image using the models that were trained by feeding pre-identified images. The last step is to interpret and show results by identifying or classifying the objects. (Sas, 2019)

Sas enumerated some of the use cases of Computer Vision:

- **"**Image segmentation partitions an image into multiple regions or pieces to be examined separately.
- Object detection identifies a specific object in an image. Advanced object detection recognizes many objects in a single image: a football field, an offensive

player, a defensive player, a ball and so on. These models use an X,Y coordinate to create a bounding box and identify everything inside the box.

- Pattern detection is a process of recognizing repeated shapes, colors and other visual indicators in images.” (Sas, 2019)

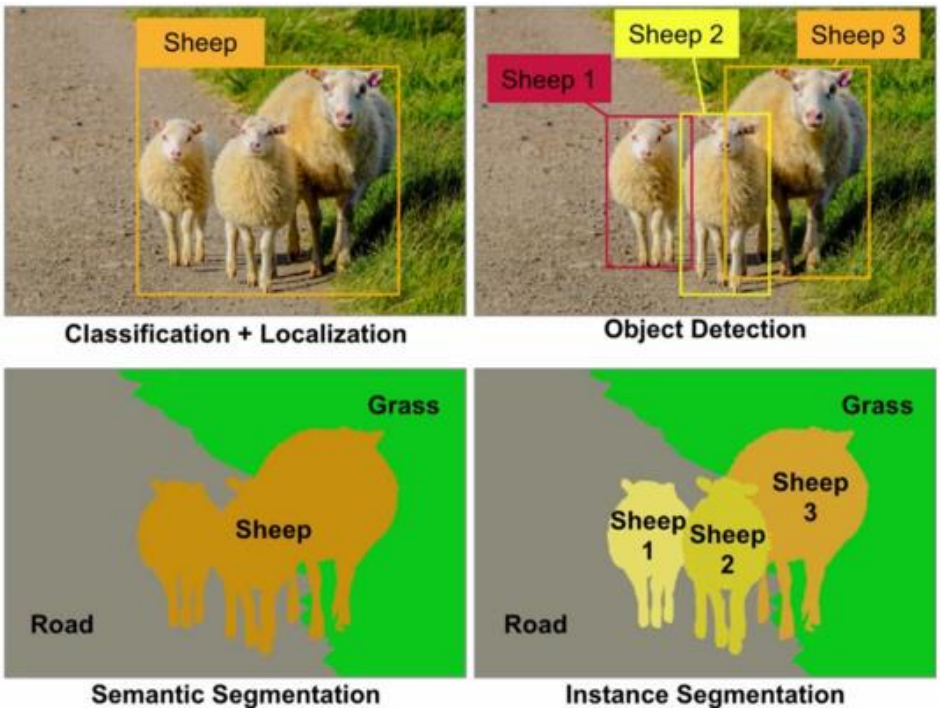


Figure 17. Image Classification and Segmentation (Venables, 2019)

Object detection is one of the use cases that is relevant to this research. It is a technology also related to image processing to locate and identify objects such as humans, vehicles, and animals, from either images or videos (Jiao, et al., 2019). This technology can classify just one or diverse objects from an image. YOLO (you only look once) is one of the fastest object detectors. YOLO divides the image into a cell, predicts if the object is enclosed in a cell, then classifies the object if there is any, and this is done in one go (Redmon, et al., 2016).

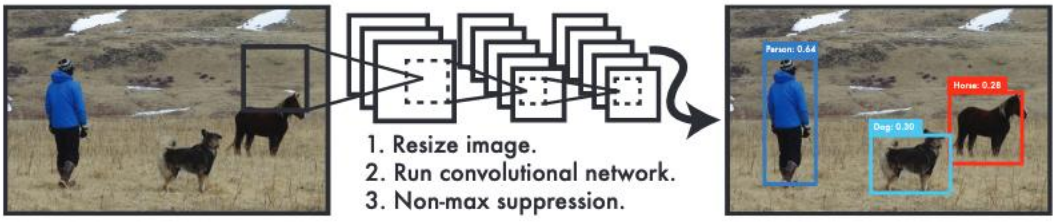


Figure 18. YOLO Detection System. (Redmon, et al., 2016)

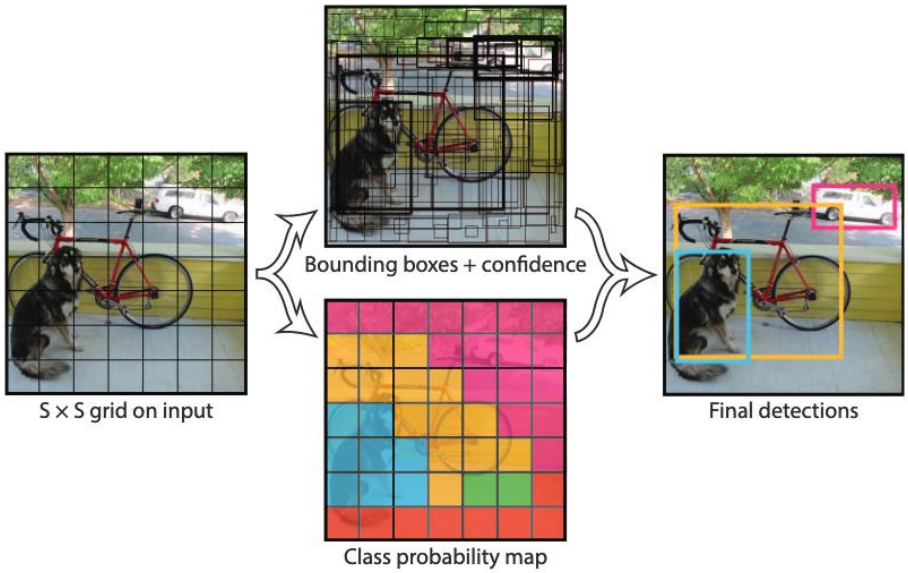


Figure 19. YOLO model. "System models detection as a regression problem" (Redmon, et al., 2016)

The practical usage of object detection in the scope of this thesis is to locate elements that display potentially localizable data. Traditional test frameworks can enumerate these elements, but they needed to access the app's elements tree. UI test frameworks such as Appium and XCUITests use element ID or XPath to locate an item in the UI. These identifiers are hardcoded in the test. The test must first find the element to perform actions such as tapping a button or sending keys to a text field. However, UI constantly changes to adhere to new UX guidelines, or new features are being added. As an example, XPaths can change by reordering view hierarchy, which means test scripts must be updated. This is just one of the reasons that make UI test automation hard to maintain and fragile, especially during active app development.

Integrating computer vision technology to the test framework can help solve this issue. CV detects objects such as UI controls and elements on the image, thus eliminates the requirement of the test framework to know the UI view hierarchy to locate a control. TechBeacon employed the same technique when they faced difficulty in implementing tests in one of their clients. Since they cannot access the element tree of the app, they use CV to detect controls on pages. They took the screenshots manually, labeled the image, and generated a metadata XML file containing the element category and their respective coordinates. These files were used to train their network for 4 hours. (TechBeacon, 2018)

User Interfaces can look different depending on the device's orientation, screen resolutions, and operating system version. The traditional UI test automation framework alone cannot validate the visual aesthetics of the app. It requires humans to inspect visually. *Applitools Eyes* is addressing this issue by using Cognitive Vision Technology. It first establishes the baseline appearance of the app per environment, and then on the next run, it will compare the differences between the screenshot and the baseline image. It uses AI-powered computer vision algorithms to detect and report only the differences that are obvious to the users. (Applitools, 2019)

Inconsistencies between the UI design and implementation is another defect that the test automation framework cannot detect. However, the research conducted by Chen et al. proved that the computer vision method could be used to identify inconsistencies in the layout, such as positions, sizes. It can also verify the presentation characteristics such as colors and fonts. Their solution - UI X-Ray achieved a “99.03% true-positive rate, which significantly surpassed the 20.92% true-positive rate obtained via manual analysis” (Chen, et al., 2017).

2.4.3 Natural Language Processing in Automated Testing

“Natural language processing (NLP) is one area of artificial intelligence using computational linguistics that provides parsing and semantic interpretation of text, which allows systems to learn, analyze, and understand human language” (IBM, ei pvm).

NLP practical applications are already used in our daily lives. Alexa, Siri, Google Translate, the Spam filtering in our mailboxes, or just by typing into the web browser’s search bar and many others. The table below lists a few applications of NLP.

Table 1. Categorized NLP applications (Hapke, et al., 2019)

Search	Web	Documents	Autocomplete
Editing	Spelling	Grammar	Style
Dialog	Chatbot	Assistant	Scheduling
Writing	Index	Concordance	Table of contents
Email	Spam filter	Classification	Prioritization
Text mining	Summarization	Knowledge extraction	Medical diagnoses
Law	Legal inference	Precedent search	Subpoena classification
News	Event detection	Fact checking	Headline composition
Attribution	Plagiarism detection	Literary forensics	Style coaching
Sentiment analysis	Community morale monitoring	Product review triage	Customer care
Behavior prediction	Finance	Election forecasting	Marketing
Creative writing	Movie scripts	Poetry	Song lyrics

An NLP processing system is often referred to as a pipeline because it usually involves several stages of processing where natural language flows in one end, and the processed output flows out the other (Hapke, et al., 2019). The two main processing steps are: “preprocessing the text and using AI to understand and generate language” (Taulli, 2019).

Cleaning and preprocessing involve tokenization, stemming, and lemmatization. During tokenization, texts are parsed and segmented in various parts. Texts are also normalized to simplify the analysis, like converting to upper or lower cases and removing punctuations. Stemming, on the other hand, is a process of removing prefixes and suffixes to extract the root word. Lemmatization then finds the similar meaning of the root word, “better” is lemmatize to “good” as an example. (Taulli, 2019)

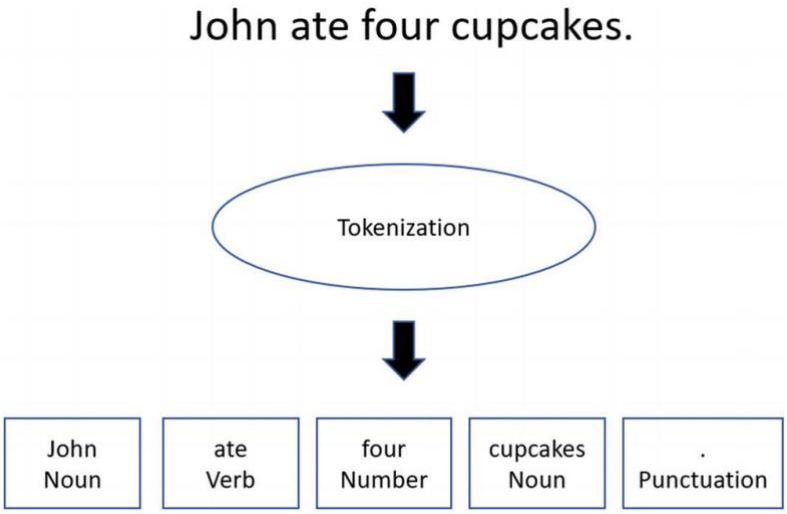


Figure 20. Example of tokenization (Taulli, 2019)

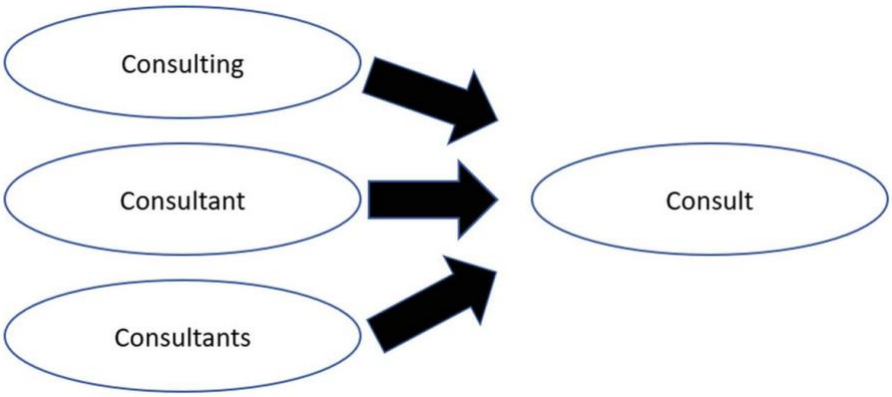


Figure 21. Example of stemming (Taulli, 2019)

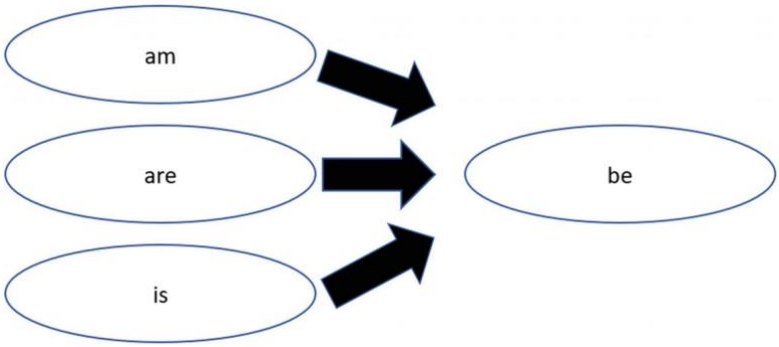


Figure 22. Example of lemmatization (Taulli, 2019)

To understand or extract information or knowledge from natural language text, researchers commonly use the following approaches:

- *Named entities and relations recognition*. A typical sentence may contain several named entities such as location, organizations, people, dates, times, and events. This is a process of identifying words that represent the mentioned entities. (Hapke, et al., 2019)
- *Part-of-speech (POS) tagging*. A process of recognizing what parts of speech do the word belong, such as verbs, adverbs, nouns, etc.
- *Topic modelling*. A process of finding hidden patterns and cluster. (Taulli, 2019)
- *Chunking*. Processing text in phrases. (Taulli, 2019)

Entity extraction is one of the useful features of NLP for UI localization testing. The ability to recognize names, dates, times, and locations have many use cases for localization. These are localizable data, thus if the test framework can extract those, then definitely it can validate that the extracted data adhere to the expected locale format.

Translation service providers have many uses of NLP. As an example, Jonker enumerated some of the applications of NLP for localizer such as (a) extracting all names before translation to make sure they are handled correctly afterward, (b) extracting key terms for glossaries, and (c) highlighting locations' geopolitical names for localization (Jonkers, 2018).

NLP is also now finding its way in *Scriptless Test Automation*. Scriptless or less coding approach in testing abstracts the underlying test code intended for manual testers that lack programming skills or stakeholders that have no technical expertise. Tools like Testsigma use NLP for test case creation and have AI at its core to allow writing of test in plain natural language, which can easily be understood (Testsigma, 2020). The example test is shown below (Lavanya, 2019).

> "Go to https://testsigma.com",
> "Enter Name in the Username field",
> "Verify that the page displays text Testsigma" (Lavanya, 2019)

3 Solution Building

The knowledge presented in the theoretical background section is the building block for the proposed solution presented in this chapter. First, the requirements are enumerated based on the current testing practices of Quality Engineers in the company that this research was conducted. Then, based on the requirements, design decisions, and implementation details of a prototype system called NEAR (**N**avigate, **E**xtract, **A**nalyze and **R**eport) are presented.

3.1 Localization Testing Requirements

Before answering the research question about the usage of AI in automated localization testing, the author, together with the Quality Engineers, evaluated if there are any steps in localization testing that are worth automating. A set of questionnaires were sent to two QE leads, two members of localization teams, one manager, and three senior quality engineers. The questions are listed in the table below:

Table 2. Questionnaires

Questions
What kind of issues do you look for when testing localization?
How often do you test localization? How much time do you allocate per test run?
Should this be automated or manual testing is enough?
Do you have a TA system to test localization? If yes, what framework/tools do you use?
If there is a prototype system to do this, are you willing to pilot to do a few test runs?

The following conclusions were made based on the gathered answers.

- Localization testing should be semi-automated.
- Manual verification by a person that knows the context should still be done to check the findings reported by the automation process.
- On average, one day is allocated every release for localization testing.
- Technologies that they are familiar with:
    - Python language
    - Selenium
    - Appium
    - Alchemy Catalyst
    - UIAutomator

Based on the discussion afterward, the following are the steps that the automated system should accomplish.

- Generate screenshots. Store these as artifacts, also draw a bounding rectangle for offending lines or words that are found in the image.
- Detect non-localized string. These can be hardcoded or placeholder texts.
- Detect overlapping strings.
- Detect truncated strings.
- Find wrong spellings.
- Find corrupted characters.
- Find dates, currencies and numbers that are not formatted according to the locale.

3.2 NEAR System Implementation

The prototype system is called NEAR which stands for **N**avigate, **E**xtract, **A**nalyze and **R**eport.

After the requirements were narrowed down, a draft of the system's design was drawn. The diagram provides an overview that even a non-technical person can understand, and it also serves as a starting point when discussing ideas with peers. The figure below illustrates how the components interact.

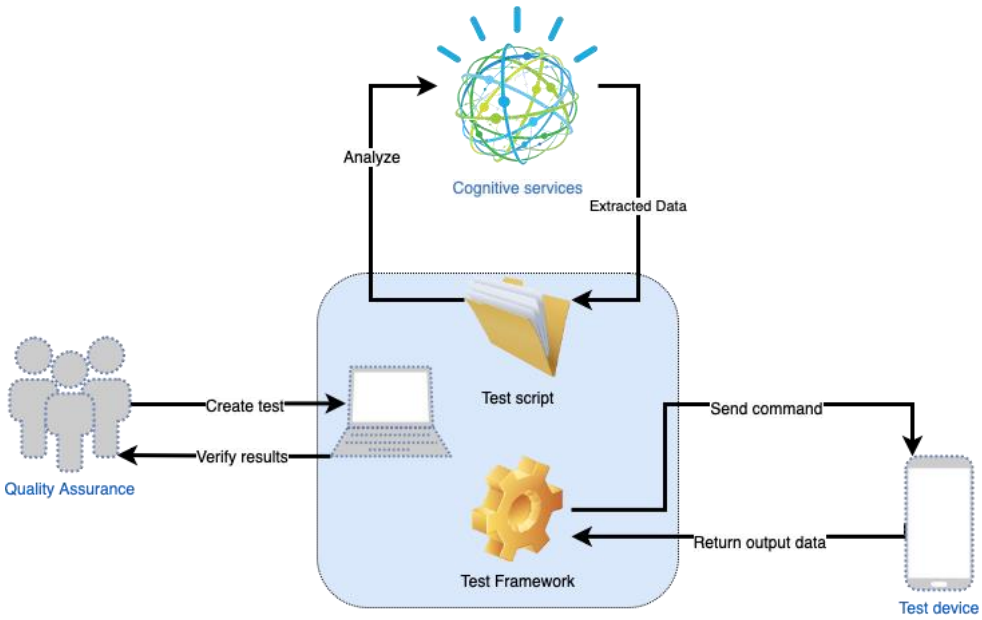


Figure 23. High level design draft

The process starts and ends with Quality assurance, a group of people composed of Quality Engineers, and at least one language expert. They are responsible for creating test scripts and validating the results. These test scripts navigate the app's UI and take screenshots. Likewise, it passes these images for further data extraction to cognitive service, a cloud-based AI platform for running computer vision and NLP algorithms. The predefined rules are then applied to the accumulated data to determine the results. The component diagram is show in the figure below.

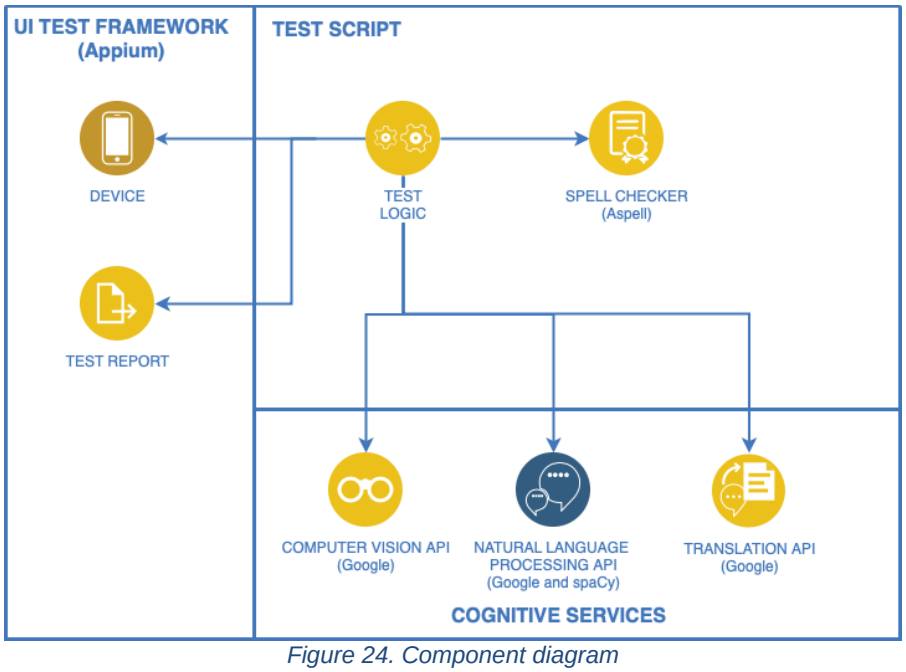


Figure 24. Component diagram

3.2.1 Development and Testing Environment

Quality Engineers will be the ones to adopt and develop this prototype further. Therefore it is imperative to align the tools with their existing skills.

Test creation. The chosen language was Python, and the automation framework was Appium. The test scripts were written using Visual Studio Code as editor. The set-up is similar to the figure shown below.

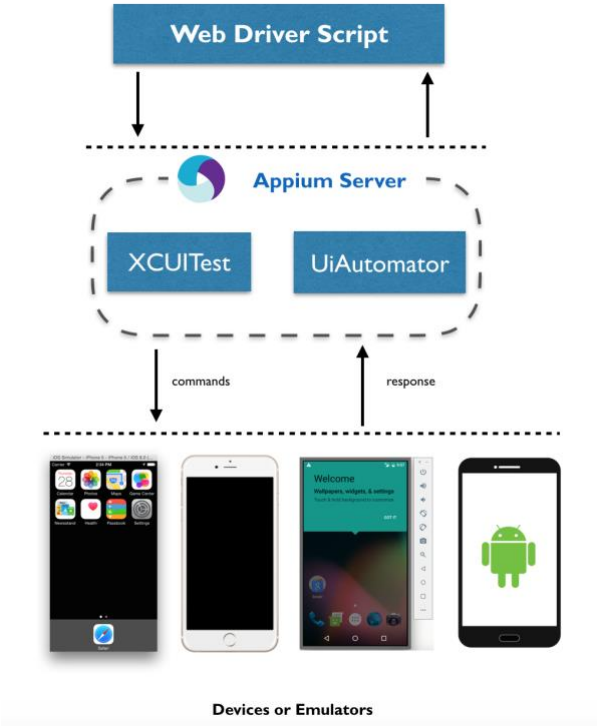


Figure 25. Appium architecture (Verma, 2017)

Test execution. Since the system is in prototype level, it was not integrated with Jenkins CI server. Tests were run on a local machine where the mobile devices are connected via USB. To support multi-platform testing, Macbook was used with XCode 11, Android Studio 3.5, and Appium 1.17.0-beta.1 installed.

Test app. An iOS app was created as a proof-of-concept. It serves as an example for App Under Test (AUT). It has UI elements that display the typical localizable data such as date, currency, and number values. Although the original intention was to localize the app in Finnish, it is merely not possible because most cloud-based AI services like Google Vision and Amazon Textract do not support this language yet. For research

purposes, the Spanish language was chosen instead. This app was adopted from the article about localization that Malliswamy wrote (Malliswamy, 2018).

The baseline UI is shown in the figure below. During development, the app was modified to introduce various kinds of localization issues in order to verify the test logic.

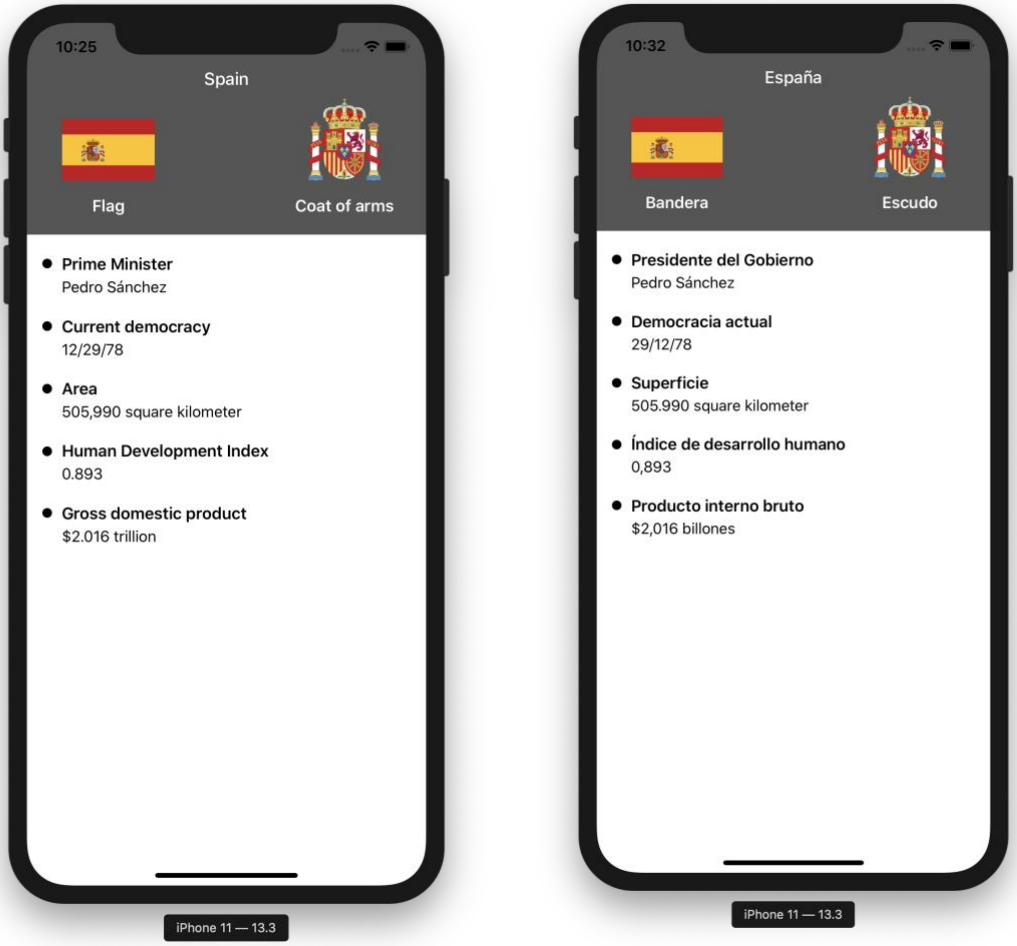
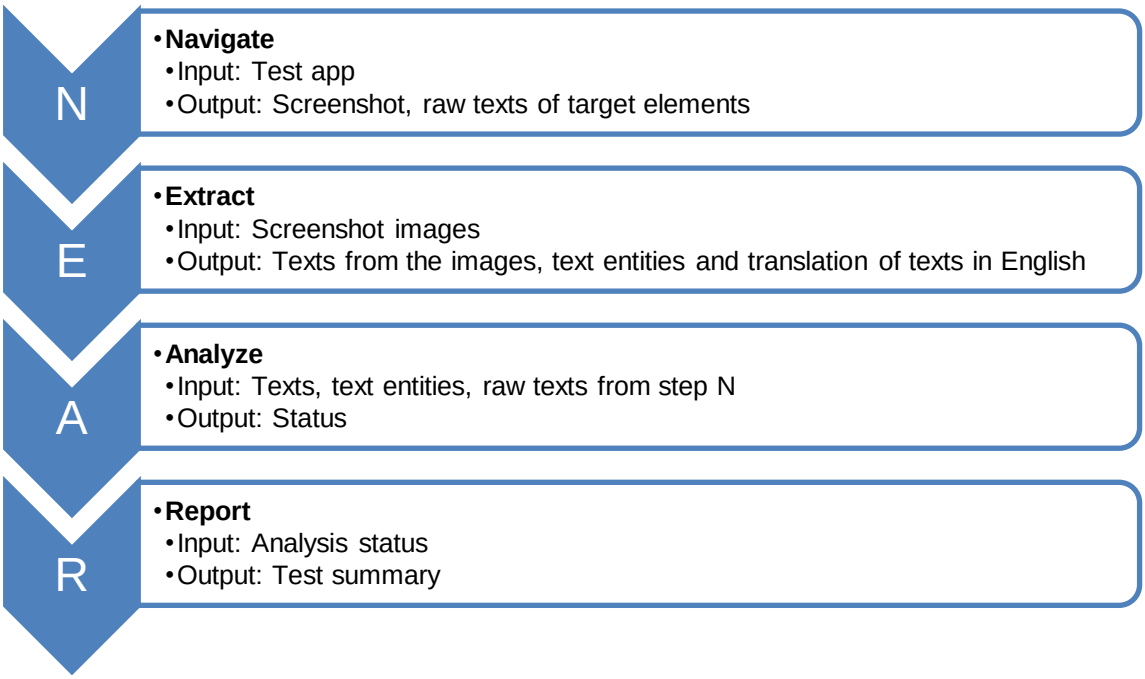


Figure 26. Test app in US and Spain locale

3.2.2 Automation Strategies

The automation approach was adapted from Ramler and Hoschek system. The steps are (a) navigating the UI, (b) extracting the UI information, (c) analyzing the extracted

data, and (d) generating a test report (Ramler & Hoschek, 2017). The input and output of each step are shown in the process diagram below.



3.2.2.1 Navigating the UI

It was decided early on to create a test tailored to the app under test instead of adopting a model-based testing approach of generating the test cases automatically. The test navigates to the UIs, but it does not verify the functionality. The element's accessibility identifier is used to locate for a specific control. Using accessibility identifier or XPath makes the test script language-independent, which means it can be reused for each localized variant of the app. The code snippet below was executed while testing both in English and Spanish.

```
def test_main_ui(self, driver, device_logger):
    screenshot_dir = device_logger.screenshot_dir

    wait = WebDriverWait(driver, 20)
    # wait for a label with 'country' as accessiblity
    wait.until(EC.visibility_of_element_located((By.ID, "country")))

    driver.save_screenshot(os.path.join(screenshot_dir, 'initial_screen.png'))

    # scroll down
    driver.execute_script('mobile: scroll', {'direction': 'down'})

    driver.save_screenshot(os.path.join(screenshot_dir, 'after_scrolling_down.png'))

    # enumerate all text elements
    captured_texts = [elem.text for elem in
driver.find_elements_by_class_name("XCUIElementTypeStaticText")]
```

Listing 6. Code snippet to wait for element with 'country' as id before scrolling and taking screenshot

The test script takes screenshots before executing navigation actions such as tapping the back and next button and scrolling. These screenshots are images in PNG format and are saved in a local folder. Likewise, the test script enumerates all the elements and string values that will be used for comparison later on. Example screenshot is shown below

Figure 27. Captured screenshot

3.2.2.2 Extracting the UI information

Appium can already extract an element's text value. However, it is not useful when validating if the text is truncated or not. It gives the entire value and not the visually visible part of the text only. This is illustrated in the figure below.

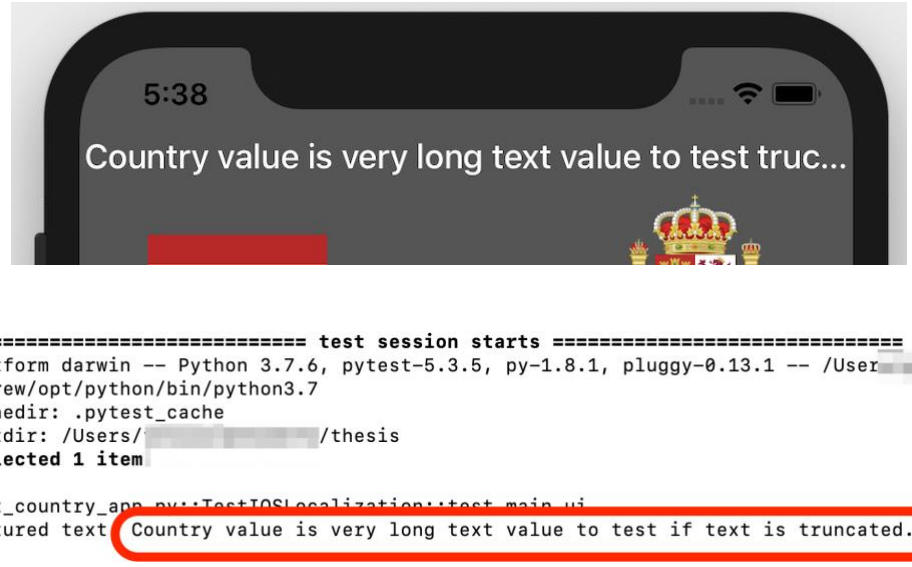


Figure 28. Visually captured text and Appium's extracted text

To capture the visually visible texts only, Vision AI was used to extract texts from images. Two cloud-based services were tested, *Google Vision* and *Amazon Textract*. Amazon Textract has a feature to extract key-value pairs, which should be very useful to retain the original context. However, it only supports the English language, as illustrated in the example screenshot below, "Sánchez" was captured as "Sanchez".

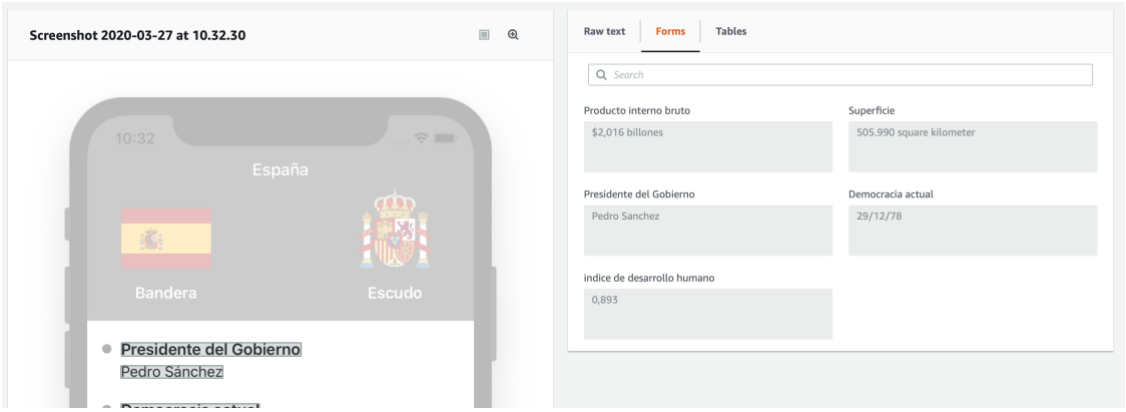


Figure 29. Amazon Textract key-value pair extraction

On the other hand, Google Vision's Optical Character Recognition (OCR) feature supports multiple languages. However, the blocking of text is somewhat inconsistent, as shown in the figure below. This impediment is slightly irrelevant for localization testing.

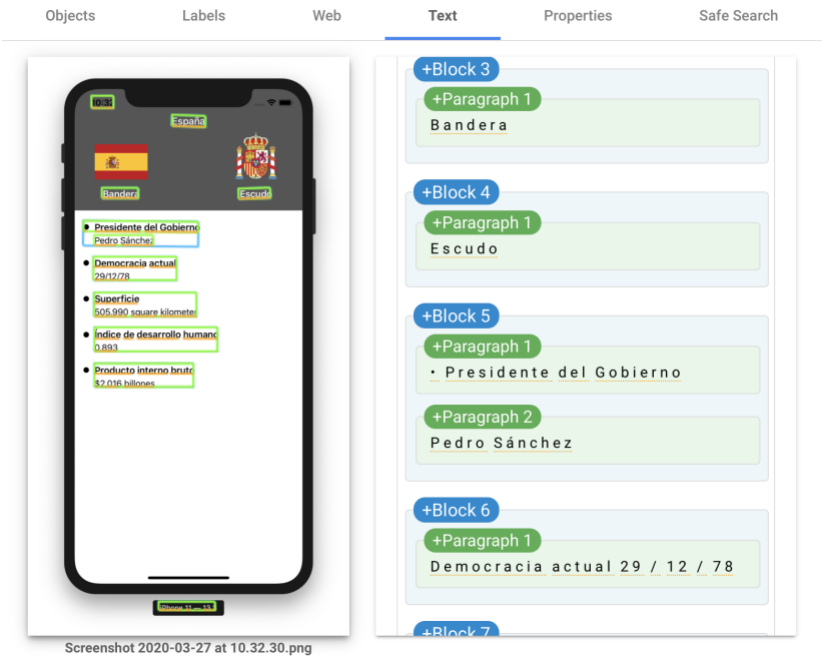


Figure 30. Google OCR with text blocking

The Google computer vision API returns paragraphs texts and vertices for the bounding rectangle. The extracted texts are then analyzed by natural language processing service to identify the entities.

Three natural language processing API was tested, IBM's Watson Natural Language Understanding, spaCy, and Google Natural Language API. All are using deep learning-powered models. spaCy is handy as it is not a cloud-based service, but rather provides pre-trained models that are available for download and install. It was able to identify most entities from the provided sample text, but it also misinterprets entities in a string like "Superficie 505.990 kilometro cuadrado", it tag *kilometro* as a *PERSON*. Google's NLP was chosen instead.

Afterward, those texts are translated into English by the cloud translation service. These values are then accumulated and stored in a dictionary data structure. The flow chart is shown below.

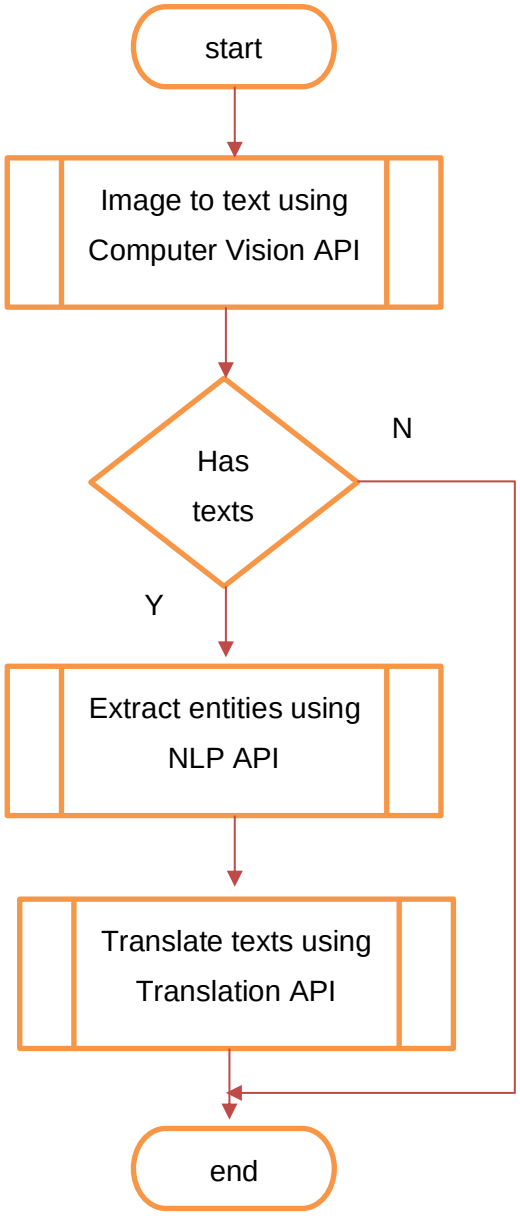


Figure 31. Flow chart for data extraction using cloud-based AI services

3.2.2.3 Analyzing the Extracted Data

The table below is tabulated data accumulated from different sources. The data serve as input to the set of rules to verify if specific criteria are met. If one of the rules returns true, it is considered a failed test.

Table 3. Tabulated data to be analyzed

Item	Description	Source
Raw texts	Array of elements text values	Appium XCUIElementTypeStaticText values.
Paragraphs	String	Computer Vision API
Paragraph bounds	Vertices of the bounding rectangle per paragraphs	Computer Vision API
Words	List of words in a paragraph	Computer Vision API
Words bounds	Vertices of the bounding rectangle per word	Computer Vision API
Entities	NLP entities found in the paragraph	Natural Language Processing API
Translation	English translation of the paragraph	Translation API

The following basic rules are applied:

Misspelling. Each word checked for misspelling using *Aspell*, an open-source spell checker.

Wrong format. An entity such as date or number is verified if it follows the locale format for displaying such data.

Untranslated string. A paragraph is compared with its English translation. If the value is the same, this means that it is a hardcoded string or invalid value. The translation service also returns the detected source language. This value must match with the current language of the system.

Truncation. Ellipsis at the end of the string usually indicates that the text is truncated. However, there might be cases that this is intentional. A list of raw text values is iterated and checked if it starts with the same value as the given paragraph. If it is, but the length is not the same, this can signify that the text is truncated.

Bounds are used to draw a rectangle in an image if a rule criterion is not met. Each rectangle color signifies a type of issue found.

3.2.2.4 Generating a Test Report

The test output is a summary of the test cases and status. If one of the test cases failed, it would be listed together with the corresponding image that has a bounding rectangle drawn on a word or paragraph that failed to satisfy the requirements.

```
$ py.test test_country_app.py -v
============================================= test session starts ==============================================
platform darwin -- Python 3.7.6, pytest-5.3.5, py-1.8.1, pluggy-0.13.1 --
cachedir: .pytest_cache
rootdir:
collected 1 item

test_country_app.py::TestIOSLocalization::test_main_ui PASSED                                          [100%]

============================================== 1 passed in 8.14s ===============================================
FSAPPLE2125:thesis $
```

Figure 32. Example output of a successful test

```
==================
[BLUE] Invalid format: 1
[RED] Untranslated: 1
[YELLOW] Truncated: 1
[GREEN] Misspelled: 1
Result image: /[redacted]r/results/2020_04_05_12_45/screenshots/out_main.png
==================
FAILED

=================================================================================== FAILURES ===
```

Figure 33. Example output of a failed test

A rectangle is drawn into a word if the spelling is wrong. If a paragraph is truncated, untranslated, or contains a number or date that is not properly, then a rectangle is drawn on the paragraph bounds. The following colors are used depending on the type of issue found.

- BLUE. Invalid date or number format.
- RED. Untranslated paragraph.
- YELLOW. Truncated paragraph.
- GREEN. Misspelled word.

Figure 34. Rectangle drawn on the truncated paragraph

4 Solution Evaluation

After the development of the prototype system, the research moved to find the apps to use for testing in order to determine the system's strengths and weaknesses. This chapter describes the testing results, observed behaviors, and recommended future enhancements.

4.1 Results

NEAR System evaluated three iOS apps with source codes that were downloaded from the internet and compiled to run on iOS 13. The apps languages were Spanish, German, and Russian. All apps contained a total of 51 localizable data, such as numbers, dates, names, and descriptions. Apps' UI elements included buttons, labels, and images.

The evaluation coverage included detecting misspelling, untranslated or hardcoded string, wrong number or time format, and truncated string. The total running test time for three apps on one platform was around 5 minutes. It averages 6 seconds to evaluate one element.

NEAR found four issues from all of the tested apps, two of those are truncated strings and two invalid number format. There were four false positives, and those are year values that are tagged as both YEAR and NUMBER entities by natural language processing service. In this particular case, two rules were used, such as validating the date format and also the number format. The system expects that 2014 must be written as 2 014 in *ru_RU* locale. The date rule should override the number rule if both are applicable. Some of the screens with detected issues are illustrated below.

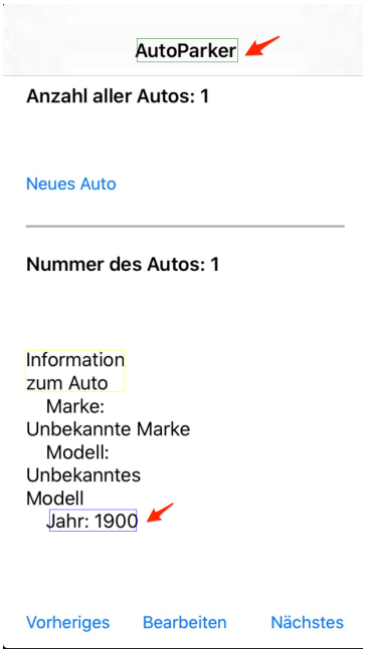


Figure 35. False positives for year and title strings. App under test is AutoParker (Sharp, et al., 2013)

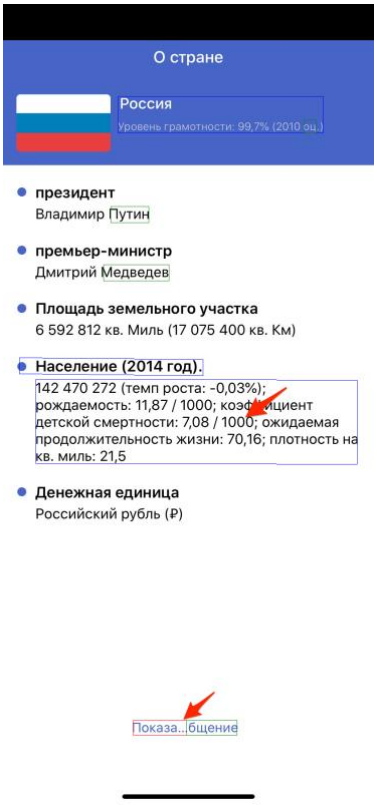


Figure 36. Truncated string and wrong number formatting. App under test is from an article written by Malliswamy (Malliswamy, 2018)

Figure 37. Invalid number format. App under test is iLikeIt (Raywenderlich, 2017)

4.2 Limitations

The system has limited language support. NLP entities feature dictates what language the NEAR system will support. As of this writing, Google Cloud Natural Language supports 11 languages.

The system's validation logic is dependent on the tools used. As an example, python's *de_DE* locale does not have a separator for thousand, but in iOS, the separator is a period character. This limitation can cause false positives when validating if a number is formatted correctly.

Another observation is with the *Aspell* spell-checker. It is necessary to exclude or white-list the numbers and names of persons because these can also generate false-positive results. This can be a burden to maintain as more and more entity types needed to be white-listed.

NEAR system uses Appium API to enumerate raw text values for each element. The output list serves as a lookup table when checking if the visually captured text is truncated or not. However, the raw strings can contain line breaks or indentations. This can result in false-positive because vision API can capture the indented text in the new

line as another element causing multiple paragraph blocks instead of just a single paragraph.

4.3 Future work

Addressing false-positives is left for future work. Caching mechanism is also needed to reduce the usage cost of cloud-based cognitive services. For efficiency, it is also recommended to only reprocess the screenshots if the captured images from the previous test run are different from the current one.

5 Summary and Conclusions

There are a few research papers that focus on automated localization testing. This thesis contributes to the practical implementation intended for testing a localized mobile app.

5.1 Summary

This thesis commenced by highlighting the significance of localizing the app. The work continued by gathering the requirements from the case company. It became apparent that the lack of an automated system for testing localization was due to the difficulties and challenges that Quality Engineers are facing while testing it. Then the research focused on finding out from the stakeholders such as Quality Engineers and fellow developers the tasks that they wish to be automated.

The theoretical background chapter then described the core concepts of the app localization process, testing strategies, and previous research conducted to automate the testing process. Due to the lack of tools designed for localization testing, existing UI automation frameworks were evaluated. The research then focused on AI subfields such as Computer Vision and Natural Language Processing, exploring the existing pre-trained models from cloud-based AI service providers such as Google, Amazon, IBM, and Microsoft. Then the prototype system called NEAR was developed. The system was then evaluated with the three iOS apps in multiple languages.

5.2 Conclusion

The prototype system proved to automate 70% of tasks enumerated from the requirements. It just took 5 minutes to test 3 apps in one platform. Considerably faster than manual testing. However, the limited language support of cloud-based NLP and Computer Vision models from service provider hindered the original intention of using the system for testing localization such as Finnish and Swedish. Nevertheless, German language support is already essential because that is one of the huge markets for the products of the case company.

This thesis answered the research question if AI can be used to improve UI localization testing. The prototype proved that it could be used, and it provides a visual context for the test, considerably faster to run and repeatable. However, it is underutilizing the capability of AI, using it only for data extraction. The data analysis was done on the python script with custom made rules. Ideally, machine learning or deep learning models should be designed and used to identify issues without the need to post-process the results. This is the side effect of relying alone on ready-made or pre-trained models from AI service providers. It is recommended to develop and train a model specific for the requirements to have a more specific context.

References

Alameer, A., Mahajan, S. & Halfond, W. G. J., 2016. *Detecting and Localizing Internationalization Presentation Failures in Web Applications.* Chicago, 016 IEEE International Conference on Software Testing, Verification and Validation .

Amazon, 2020. *Amazon Rekognition.* [Online]
Available at: https://aws.amazon.com/rekognition/
[Accessed 7 April 2020].

Amazon, 2020. *AWS Device Farm.* [Online]
Available at: https://aws.amazon.com/device-farm/
[Accessed 6 March 2020].

Amazon, 2020. *What Is Amazon Textract?.* [Online]
Available at: https://docs.aws.amazon.com/textract/latest/dg/what-is.html
[Accessed 7 April 2020].

Amazon, 2020. *What is Computer Vision ?.* [Online]
Available at: https://aws.amazon.com/computer-vision/
[Accessed 6 April 2020].

Android Developer Guide, 2019. *Test your app with pseudolocales.* [Online]
Available at: https://developer.android.com/guide/topics/resources/pseudolocales
[Accessed 1 March 2020].

Android, 2019. *Automate user interface tests.* [Online]
Available at: https://developer.android.com/training/testing/ui-testing
[Accessed 4 March 2020].

Android, 2019. *Localize your app.* [Online]
Available at: https://developer.android.com/guide/topics/resources/localization
[Accessed 10 January 2020].

Anon., 2019. *Test your app in each language to ensure a successful launch.* [Online]
Available at: https://developer.android.com/distribute/best-practices/launch/test-language
[Accessed 02 March 2020].

Appium, 2019. *Appium Sample Code.* [Online]
Available at: https://github.com/appium/appium/tree/master/sample-code
[Accessed 6 March 2020].

Apple, 2015. *About Internationalization and Localization.* [Online]
Available at:
https://developer.apple.com/library/archive/documentation/MacOSX/Conceptual/BPInternational/Introduction/Introduction.html#//apple_ref/doc/uid/10000171i
[Accessed 10 January 2020].
Apple, 2015. *Localizing Your App.* [Online]
Available at:
https://developer.apple.com/library/archive/documentation/MacOSX/Conceptual/BPInternational/LocalizingYourApp/LocalizingYourApp.html#//apple_ref/doc/uid/10000171i-CH5-SW1
[Accessed 28 February 2020].
Applitools, 2019. *Applitools Eyes: Introduction to Automated Visual UI Testing.* [Online]
Available at: https://applitools.com/blog/applitools-eyes-introduction-to-automated-visual-ui-testing
[Accessed 14 March 2020].
Archana, J., Chermapandan, S. R. & Palanivel, S., 2013. *Automation framework for localizability testing of internationalized software.* Chennai, 2013 International Conference on Human Computer Interactions (ICHCI).
Arnatovich, Y. L., Ngo, M. N., Kuan, T. H. B. & Soh, C., 2016. *Achieving High Code Coverage in Android UI Testing via Automated Widget Exercising.* Hamilton, IEEE, pp. 193-200.
Atlassian, 2020. *What is automated testing?.* [Online]
Available at: https://www.atlassian.com/continuous-delivery/software-testing/automated-testing
[Accessed 6 March 2020].
Awwad, A. M. A. & Slany, W., 2016. Automated Bidirectional Languages Localization Testing forAndroid Apps with Rich GUI. *Mobile Information Systems,* Volume 2016.
Bezmolna, Victoria, 2019. *Appium vs. Espresso: Which Framework to Use for Automated Android Testing.* [Online]
Available at: https://bitbar.com/blog/appium-vs-espresso-which-framework-to-use-for-automated-android-testing/
[Accessed 5 March 2020].
Bitbar, 2019. *Client-side vs Cloud-side execution.* [Online]
Available at: https://www.youtube.com/watch?v=gz58N1vRLb8
[Accessed March 6 2020].

Capgemini, Sogeti, 2019. *World Quality Report 2019-20.* [Online]
Available at: https://www.capgemini.com/fi-en/research/world-quality-report-2019-20/#
[Accessed 4 March 2020].
Carmi, A., 2019. *Taking the Pain Out of UI Localization Testing.* [Online]
Available at: https://applitools.com/blog/taking-the-pain-out-of-ui-localization-testing-1?utm_referrer=https%3A%2F%2Fwww.google.com%2F
[Accessed 6 Februrary 2020].
Chen, C.-F.et al., 2017. *UI X-Ray: Interactive Mobile UI Testing Based on Computer Vision.* Limassol, 22nd International Conference on Intelligent User Interfaces.
Engineer, Q., 2020. *UI Testing* [Interview] (February 2020).
Facebook, 2016. *The mobile device lab at the Prineville data center.* [Online]
Available at: https://engineering.fb.com/data-center-engineering/the-mobile-device-lab-at-the-prineville-data-center/
[Accessed 6 March 2020].
Flanagan, D., 2002. *Java in a Nutshell.* 4th Edition ed. s.l.:O'Reilly & Associates.
Garg, S., 2016. *Appium Recipes.* s.l.:Apress.
Google, n.d. *Natural Language.* [Online]
Available at: https://cloud.google.com/natural-language
[Accessed 7 April 2020].
Google, n.d. *Translation.* [Online]
Available at: https://cloud.google.com/translate
[Accessed 7 April 2020].
Gundepuneni, M. et al., 2012. *Generating Localized User Interfaces.* United States of America, Patent No. 20140006004.
Haller, K., 2013. Mobile Testing. *ACM SIGSOFT Software Engineering Notes,* 38(6), p. 4.
Hans, M., 2015. *Appium Essentials.* s.l.:Packt Publishing.
Hapke, H., Howard, C. & Lane, H., 2019. *Natural Language Processing in Action.* s.l.:Manning Publications.
Hardy, C. et al., 2012. Internationalization and Localization. In: *Professional Android Programming with Mono for Android and .NET/C#.* s.l.:Wrox.
Hevner, A. R., March, S. T., Park, J. & Ram, S., 2001. Design science in information systems research. *MIS Quarterly,* 28(1), pp. 75-105.
IBM, 2016. *The future is all cloud and AI.* [Online]
Available at: https://www.ibm.com/blogs/cloud-computing/2016/12/08/future-cloud-ai/
[Accessed 27 March 2020].

IBM, n.d. *Build apps with natural language processing.* [Online]
Available at: https://www.ibm.com/watson/natural-language-processing
[Accessed 6 April 2020].
ISO9241-210, 2019. *Ergonomics of human-system interaction — Part 210: Human-centred design for interactive systems.* s.l.:ISO.
Jiao, L. et al., 2019. A Survey of Deep Learning-based Object Detection. *IEEE Access,* Volume 7, pp. 128837-128868.
Jonkers, 2018. *Applying Natural Language Processing to Localization.* [Online]
Available at: https://www.jonckers.com/applying-natural-language-processing/
[Accessed 16 March 2020].
Kukushkina, N., 2019. *How Facebook, Apple, Microsoft, Google, and Amazon are investing in AI.* [Online]
Available at: https://hackernoon.com/how-facebook-apple-microsoft-google-and-amazon-are-investing-in-ai-f58b5706e34a
[Accessed 04 February 2020].
Lavanya, 2019. *Smart Test Automation using NLP.* [Online]
Available at: https://dev.to/lvnya_c/smart-test-automation-using-nlp-h9b
[Accessed 16 March 2020].
Malliswamy, B., 2018. *Localize your Apps to Support Multiple Languages — iOS Localization.* [Online]
Available at: https://medium.com/swift-india/localize-your-apps-to-support-multiple-languages-ios-localization-ac7b612dbc58
[Accessed 24 March 2020].
Microsoft, 2016. *Use UI Automation To Test Your Code.* [Online]
Available at: https://docs.microsoft.com/en-us/visualstudio/test/use-ui-automation-to-test-your-code?view=vs-2015&redirectedfrom=MSDN
[Accessed 4 March 2020].
Microsoft, 2019. *Xamarin.UITest.* [Online]
Available at: https://docs.microsoft.com/en-us/appcenter/test-cloud/uitest/
[Accessed 5 March 2020].
Mischinger, Sarah, 2019. *Appium vs. XCUITest for Automated iOS Testing.* [Online]
Available at: https://bitbar.com/blog/appium-vs-xcuitest-for-automated-ios-testing/
[Accessed 5 March 2020].
Morgado, I. C. & Paiva, A. C. R., 2015. *The iMPAcT Tool: Testing UI Patterns on Mobile Applications.* Lincoln, NE, IEEE, pp. 876-881.

pepgotesting, n.d. *Automated software testing in Continuous Integration (CI) and Continuous Delivery (CD).* [Online]
Available at: https://pepgotesting.com/continuous-integration/
[Accessed 6 March 2020].
Peters, J. F., 2017. *Foundations of Computer Vision.* Winnipeg: Springer International Publishing.
Ramler, R. & Hoschek, R., 2017. *How to Test in Sixteen Languages? Automation Support for Localization Testing.* Tokyo, 2017 IEEE International Conference on Software Testing, Verification and Validation.
Raywenderlich, 2017. *Internationalizing Your iOS App: Getting Started.* [Online]
Available at: https://www.raywenderlich.com/250-internationalizing-your-ios-app-getting-started
[Accessed 4 April 2020].
Reaktor, University of Helsinki, 2018. *Elements of AI.* [Online]
Available at: https://course.elementsofai.com
[Accessed 10 March 2020].
Redmon, J., Divvala, S., Girshick, R. & Farhadi, A., 2016. *You Only Look Once: Unified, Real-Time Object Detection.* Las Vegas, 2016 IEEE Conference on Computer Vision and Pattern Recognition.
Sas, 2019. *Computer Vision.* [Online]
Available at: https://www.sas.com/en_us/insights/analytics/computer-vision.html
[Accessed 14 March 2020].
Sas, 2019. *Natural Language Processing (NLP).* [Online]
Available at: https://www.sas.com/en_us/insights/analytics/what-is-natural-language-processing-nlp.html
[Accessed 15 March 2020].
SauceLabs, 2018. *Real Mobile Devices for Continuous Testing.* [Online]
Available at: https://saucelabs.com/sauce-labs/white-papers/real-mobile-devices-for-continuous-testing.pdf
[Accessed 6 March 2020].
Sharp, M., Sadun, E. & Strougo, R., 2013. *Learning iOS Development: A Hands-on Guide to the Fundamentals of iOS Programming.* s.l.:Addison-Wesley Professiona.
Sogeti, Capgemini, Micro Focus, 2017. *World Quality Report 2017–18.* [Online]
Available at: https://www.sogeti.com/globalassets/global/downloads/testing/wqr-2017-2018/wqr_2017_v9_secure.pdf
[Accessed 13 March 2020].

Sten Pittet, 2020. *What are the differences between continuous integration, continuous delivery, and continuous deployment?.* [Online]
Available at: https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment
[Accessed 6 March 2020].
Taulli, T., 2019. *Artificial Intelligence Basics: A Non-Technical Introduction.* s.l.:Apress.
TechBeacon, 2018. *How to use computer vision for your test automation.* [Online]
Available at: https://techbeacon.com/app-dev-testing/how-use-computer-vision-your-test-automation
[Accessed 14 March 2020].
Testim, 2018. *How AI is Changing the Future of Software Testing.* [Online]
Available at: https://www.testim.io/blog/ai-transforming-software-testing/
[Accessed 12 March 2020].
Testsigma, 2020. *Natural Language Processing (NLP)Based Test Automation.* [Online]
Available at: https://testsigma.com/blog/natural-language-processing-nlp-based-test-automation/
[Accessed 16 March 2020].
V2Soft, 2018. *Cloud based AI Services: The gateway to Artificial Intelligence in your business.* [Online]
Available at: https://www.v2soft.com/blogs/cloud-based-ai-services-gateway-to-artificial-intelligence-in-your-business
[Accessed 27 March 2020].
w3c, 2005. *Localization vs. Internationalization.* [Online]
Available at: https://www.w3.org/International/questions/qa-i18n
[Accessed 28 February 2020].
Venables, M., 2019. *An Overview of Computer Vision.* [Online]
Available at: https://towardsdatascience.com/an-overview-of-computer-vision-1f75c2ab1b66?gi=580f2b2d77a3
[Accessed 14 March 2020].
Verma, N., 2017. *Mobile Test Automation with Appium.* s.l.:Packt Publishing.
Zhao, C., He, Z. & Zeng, W., 2010. *Study on International Software Localization Testing.* Wuhan, 2010 Second World Congress on Software Engineering.