TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

KATJA KARSIKAS

TEST AUTOMATION FOR NAMED ENTITY RECOGNITION SYSTEM

Master of Science Thesis

Examiner: prof. Hannu-Matti Järvinen
Examiner, and topic approved by the Faculty Council of the Faculty of Computing and Electrical Engineering on 28 March 2018

ABSTRACT

KATJA KARSIKAS: Test automation for named entity recognition system
Tampere University of technology
Master of Science Thesis, 59 pages
October 2018
Master's Degree Program in Information Technology
Major: Software Engineering
Examiner: Professor Hannu-Matti Järvinen

Keywords: software testing, test automation, AI, named entity recognition

Software testing is an essential part of the software development process. It is needed to ensure the quality of software. As the software development process is changing towards continuous integration and deployment, an increase in the level of automation in testing is needed to ensure software quality all the time.

Using artificial intelligence (AI) creates additional challenges for testing. For instance, it can be challenging to determine whether the test output is unequivocally correct or not. In some cases, the tester can determine that AI software works correctly by using his own judgment or by getting the information from the product owner or the customer. However, in more challenging testing tasks with test automation, it needs to be very accurately specified what the correct output is and what is not.

Named entity recognition is a subcategory of AI. It means aiming to find and classify named entities in text into pre-defined categories like person and organization names. In cases of that kind, the correct outputs can be determined by people. People are needed to mark the wanted entities in text so that the automated tests can compare the output of the software with those correct entities to determine whether the software works correctly. Regardless, it is a challenging task due to different written forms of names like with abbreviations or initials. For example, initials can be used instead of a person's first or last name or in between those.

In this thesis, test automation is implemented for a named entity recognition system that recognizes person and organization names from text. It is not expected that all names are found in the text, but it is required that there are enough correct and few enough wrong names to reach the desired quality. Before the actual test automation implementation can be done, test files that match the real use cases need to be collected. In addition, the wanted outputs for the files need to be determined by marking person and organization names. This requires plenty of human work because multiple test files are needed to get statistically significant results. It also needs to be solved how to consider different forms of writing of the names. When the necessary groundwork is done, the test automation is implemented.

TIIVISTELMÄ

KATJA KARSIKAS: Nimentunnistusjärjestelmän testauksen automatisointi
Tampereen teknillinen yliopisto
Diplomityö, 59 sivua
Lokakuu 2018
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Ohjelmistotuotanto
Tarkastaja: professori Hannu-Matti Järvinen

Avainsanat: ohjelmistotestaus, testausautomaatio, tekoäly, nimientunnistus

Ohjelmistotestaus on oleellinen osa ohjelmistokehitysprosessia. Sitä tarvitaan varmistamaan ohjelmiston laatu. Ohjelmistokehitysprosessin muuttuminen kohti jatkuvaa integraatiota ja toimitusta pakottaa lisäämään automaation määrää testauksessa, jotta ohjelmiston laatu pystytään varmistamaan koko ajan.

Jos ohjelmisto käyttää tekoälyä (AI), tuo se lisähaastetta ohjelmiston testaukseen. Tällaisessa tapauksessa saattaa olla haastavaa määrittää yksiselitteisesti, onko tekoälyohjelmiston antama suorite oikein vai ei. Joissain tapauksissa testaaja voi sen tehdä käyttäen omaa harkintaa tai selvittämällä sen muuten, esimerkiksi projektivastaavalta tai asiakkaalta. Testiautomaatiota kehitettäessä tarvitsee määritellä erittäin tarkasti, mikä on oikein. Tämä aiheuttaa lisähaastetta kehittämiseen.

Nimentunnistus on yksi tekoälyn alakategorioista. Sitä käytetään etsimään ja luokittelemaan nimetyt kohteet tekstistä ennalta määriteltyihin kategorioihin, esimerkiksi ihmisen tai organisaation nimiin. Tällaisessa tapauksessa ihminen voi määritellä, mikä on oikea vastaus. Jotta voidaan testiautomaatiossa määritellä, onko ohjelmiston antama vastaus oikein, täytyy ihmisen merkitä halutut vertailuarvot tekstistä, jotta ohjelmiston antamia vastauksia voidaan vertailla niihin. Lisähaastetta tähän aiheuttavat nimien erilaiset kirjoitusmuodot, kuten lyhenteiden tai nimikirjainten käyttö. Esimerkiksi nimikirjaimia voidaan käyttää henkilön etu- ja sukunimen välissä tai korvaamassa toinen niistä.

Tässä diplomityössä toteutetaan testiautomaatio nimentunnistusjärjestelmälle, joka tunnistaa ihmisten ja organisaatioiden nimiä tekstistä. Järjestelmän ei oleteta löytävän tekstistä kaikkia nimiä, mutta sen tulisi löytää riittävän monta oikeaa ja mahdollisimman vähän vääriä nimiä, jotta sen laatuaste on halutulla tasolla. Ennen testiautomaation toteutusta tarvitsee kerätä mahdollisimman hyvin todellista käyttötapausta vastaavia testitiedostoja, joista tarvitsee määritellä järjestelmältä halutut arvot merkitsemällä niistä henkilöiden ja organisaatioiden nimet. Jotta testiautomaatiolta saadaan tilastollisesti merkittäviä tuloksia, tarvitaan testitiedostoja useita, mikä taas aiheuttaa paljon ihmistyötä. Tarvitsee myös ratkaista, kuinka nimien eri kirjoitusmuodot otetaan huomioon vertailtaessa saatua ja haluttua arvoa keskenään, jotta vertailu antaisi mahdollisimman oikean tuloksen. Kun tarvittava pohjatyö on suoritettu, toteutetaan itse testiautomaatio.

PREFACE

This thesis will conclude my master's studies at the Tampere University of Technology. I would like to thank all people who have encouraged and supported me in my studies and making this thesis.

Especially, I would like to thank Minna Vallius for giving me this subject and the opportunity to make this thesis at M-Files, supporting me through the whole process and reviewing the thesis. I would also thank my examiner Professor Hannu-Matti Järvinen for his guidance with my thesis. Finally, I would like to extra thanks to my husband Mika who helped and supported me through my studies and reviewing the thesis. Without him, I would not be here where I am now.

Tampere, 11.10.2018

Katja Karsikas

CONTENTS

LIST OF SYMBOLS, AND ABBREVIATIONS

ACE	Automatic Content Extraction
AI	Artificial Intelligence
API	Application programming interface
AUC	Area Under the Curve
CSV	Comma-separated values
DevOps	A practice to unify software development (Dev) and operation (Ops)
FN	False negative
FP	False positive
GUI	Graphical user interface
IML	Intelligent Metadata Layer
JSON	JavaScript Object Notation, lightweight data-interchange format
NER	Named entity recognition
NLP	Nature language processing
OCR	Optical character recognition
OS	Operating system
PDF	Portable document format
ROC	Receiver Operating Characteristic
TN	True negative
TP	True positive
UI	User interface

1. INTRODUCTION

Software testing is an important part of the software development process. Purpose of software testing is to create information about the quality of a software product (ISO/IEEE/IEC, 2013). It is defined as the process of systematically searching faults from the executable software (Haikala & Mikkonen, 2011, p. 205). Being able to uncover faults from the product will then help improve its quality. Testing can be done on many levels, and its coverage mainly depends on the amount of it.

In current software development practices software is developed in shorter development cycles using methods like continuous integration and delivery. To do all this work in a solid manner, a significant amount of automation and monitoring is needed. Therefore, all the steps of software construction from integration, testing and releasing to deployment and infrastructure management are to be automated as much as possible.

Artificial intelligence (AI) is used to solve problems that would otherwise be difficult to solve just by writing an algorithm for them. An AI system is only told what kind of output is needed and the system itself tries to figure out a way to produce it. Due to the nature of programs using AI, it might be difficult to define the correct outputs, which creates extra challenges also for the testing of those programs. If for example, a system picks only keywords from a text, the correctness of the output is only a subjective view of the relevant content. For example, three people could all define different keywords. This causes challenges, especially when developing test automation since the correct outputs need to be defined accurately for the machine to be able to determine whether the output is correct or not.

Named entity recognition is a subtask of information extraction that is one of the subcategories of artificial intelligence. In named entity recognition, the goal is to find and classify named entities in text into pre-defined categories like a person name, an organization name, an email address, and locations. In this kind of case, the correct outputs can be determined by people. People need to mark the wanted entities in text so that the test automation can compare the output of the software with those marked as correct entities to determine if the software works correctly. Different forms of writing cause some challenges for comparison tasks in test automation. Organization names can, for example, be written as Auto By Cat Corporation, Auto By Cat Corp., Auto By Cat, or ABC. Person names can also be in different formats. For example, Maurice R. "Hank" Greenberg, Maurice R. Greenberg, Maurice Greenberg, M. Greenberg or Dr. Greenberg.

Purpose of this master thesis is to implement testing automation for an application which extracts named entities such as person or organization names from documents. The application is a part of a greater software system and it uses artificial intelligence (AI) to recognize the wanted entities. Due to the currently used software development practices, software has a short release cycle and the whole software system as well as its applications should be in good quality all the time. Test automation is needed to ensure that the system always works with the latest software version.

First, a literature review to related topics is done. One of the topics is software testing. It is described what is the role of testing in the software development process and what software testing is generally focusing on from an automation aspect. Another topic is DevOps that is software engineering culture and practice. The third topic is artificial intelligence (AI). It is defined what AI means, what is the development process of systems that uses AI and how systems that use AI should be tested. The last topic is M-Files as a company and their product.

After the literature review, the focus will be on the problem that is this thesis is all about. The problem is described in more detail and different challenges related to it and their solutions are gone through. Lastly, the implemented solution to the problem is presented and explained and it is described how it could be developed in the future.

The thesis is divided into five chapters. Chapter two gives more detailed background information about related topics. Chapter three gives more specific information on the problem. Chapter four describes the solution of the implemented test automation. In chapter five, different alternatives for developing the implementation forward are considered. The last chapter concludes all that has been done in this thesis.

2. BACKGROUND

This chapter gives background information about related topics to the thesis. The goal of the thesis is to implement test automation for the part of M-Files product that uses artificial intelligence solution. Therefore, the basic concepts of software testing, artificial intelligence, and M-Files as a product are introduced. In addition, DevOps software engineering culture and M-Files as a company are presented because those are the essential reasons for this thesis.

2.1 Software testing

2.1.1 Software testing in the development process

The main goal of software development is usually to produce a software product based on client requirements. This regularly includes multiple phases: specifying, designing, programming, testing, quality assurance, and documentation. The waterfall model has been used as the general software development model for a long time. In the waterfall model, the design and development processes proceed downwards in phases as can be seen in Figure 1. In this model, testing is only a phase between coding and operations.

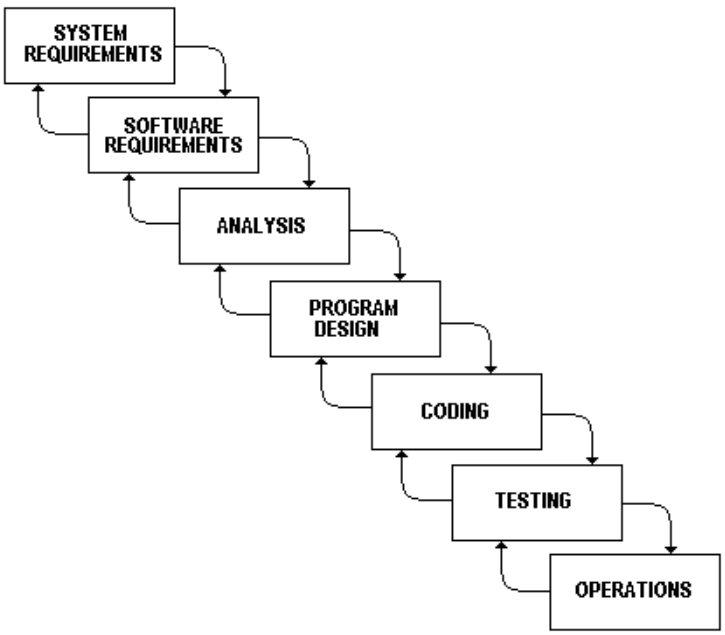

Figure 1. *Waterfall model. (Royce, 1970)*

Currently, the waterfall approach is considered old-fashioned in many aspects (Kasurinen, 2013, p. 14). In that model, the testing starts only after all the planning, and development work has been completed. However, there also exist development models

that integrate development, and testing, and are therefore considered more efficient. An example of such a model is the V-model that is demonstrated in Figure 2. The model can be thought of having two sides, software development, and quality assurance, in the shape of letter V.

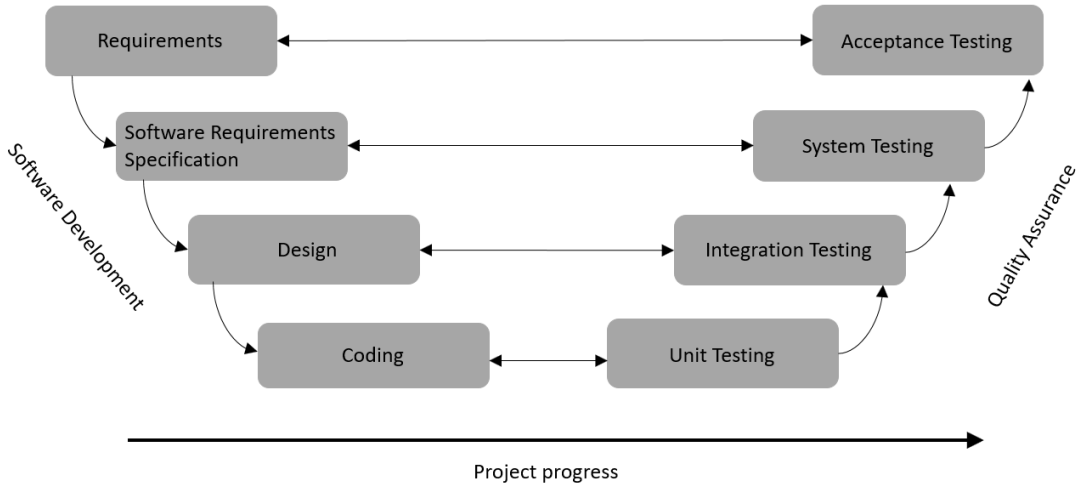

Figure 2. *V-model (Kasurinen, 2013, p. 14).*

In the V-model, testing is planned during the development process and conducted during the quality assurance part i.e. the right side of the V. There are specific testing methods to ensure the quality of the product for each planning phase. This means that programming work is inspected by unit tests, the realization of the plan by integration tests, the accuracy with the specification by system tests, and general requirements by separate acceptance tests. If the program passes all phases and their respective tests, it is accepted, and ready for use. (Kasurinen, 2013, p. 14)

Recently, an agile software development process has started to spread and has started to replace the waterfall approach (Haikala & Mikkonen, 2011, p. 46). An agile development process is an iterative and incremental approach to developing software products. From the software testing perspective, this means that the testers test each increment of coding as soon as it is finished. In the agile development process, the work proceeds based on user stories that are used to create simplified descriptions of the product requirements. The idea of a user story is to describe a specific software feature from an end-user perspective. Now in the agile model, programmers never get ahead of the testers because a user story is not considered “done” until it has been tested. (Crispin & Gregory, 2009, p. 12)

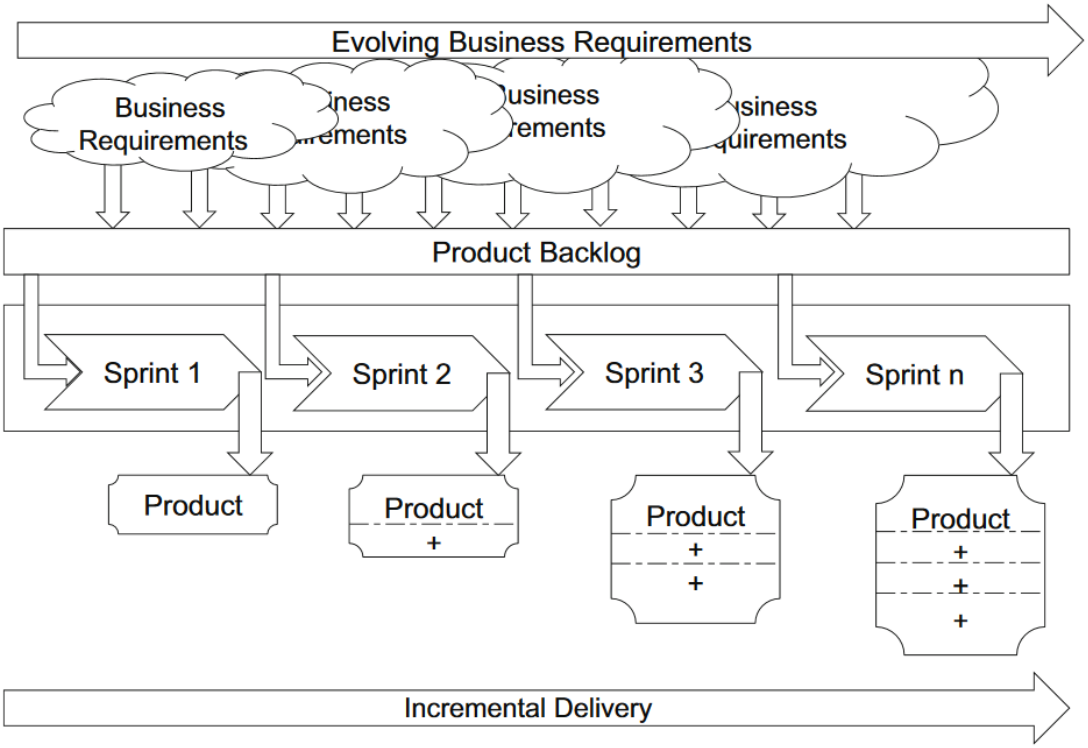

Figure 3. *Example Scrum project lifecycle (ISO/IEEE/IEC, 2013, p. 38).*

A popular agile method is called Scrum (Figure 3). In the scrum method, a software project progresses in a series of sprints. Each sprint normally results in a new functionality that can be delivered to the users. The new functionality can be an addition to existing functionality or a completely new functionality. A sprint lasts typically 2-4 weeks and includes designing, coding, and testing of the features that were meant to be implemented during the sprint. Before each sprint, a planning meeting is arranged, and it is decided which items from the product backlog, which includes customer requirements as user stories, can be managed during it. Those items are then gathered as a sprint backlog and implemented by the development team during the sprint. At the end of the sprint, all backlog items designated for that sprint should be ready for production. This process makes it potentially possible to ship the product at the end of each sprint. (ISO/IEEE/IEC, 2013, p. 38)

2.1.2 Software testing in general

Software testing means the work that is done to ensure that the implemented software product fulfills all the requirements set for it and that all included features work as they are meant to. In other words, it is ensured that the product has been done right and that it is the right product. (Kasurinen, 2013, p. 10)

Software testing is an essential and necessary phase in the software development process. It is defined as the process of systematically searching faults from the executable software. Being able to uncover faults from the product will then help improve its quality.

On the other hand, software testing cannot be used to ensure that the product is completely error-free, and, hence, it cannot be used to show that the product is full-proof. Testing can be done on many levels, and its coverage mainly depends on the amount of it. The product development always includes the question of how much testing should be done, since it is a compromise between resources (time, money, and equipment etc.) and the desired level of operating reliability. (Haikala & Mikkonen, 2011, p. 205)

The terms error, bug, fault, mistake, and defect are commonly used as synonyms in software testing even though they mean slightly different things. A fault is considered as the actual cause for an error which is used to describe an action that is either inaccurate or incorrect. In the context of testing, an error is a deviating function or an end result from the product specification. Logical testing without a defined specification is therefore impossible. There can be several different levels for an error from mild, or insignificant to serious. The seriousness of an error can vary from a cosmetic detail to failure which blocks the usage of the entire software. The errors found during the testing processes need to be reported and analyzed so that they can be fixed. (Haikala & Mikkonen, 2011, pp. 205-206, 216)

The software testing process itself can also be divided into phases that are designing of the testing process, creating the test environment, conducting the tests, and analyzing the obtained results. Designing the testing process includes a number of other things such as creating the test plan and selection of the test cases. The test plan describes what kind of tests are run, and when, as well as, what are the expected results, and when testing is finished. There are two ways of selecting test cases: white box or black box testing. In the black box testing, the test cases are selected based on specifications without seeing the actual implementation. On the other hand, in the white box testing, the implementation is known and used when selecting the test cases. (Haikala & Mikkonen, 2011, pp. 205-216)

The software testing has different levels from component testing to the whole system testing. When an error or a fault has been discovered, the higher the testing level the higher the expenses caused by fixing it (Haikala & Mikkonen, 2011, p. 208). Therefore, it is desirable to try to find as much of the errors and faults as possible in the lower level tests. Unit tests and component tests are among the lowest level of tests. A unit test verifies the behavior of a small subset of the system, such as an object or a method. A component test verifies the behavior of a larger part of the system, such as a group of classes that provide some service. When moving higher in the testing levels, there are the integration tests. In integration testing, the interaction between selected components is tested in addition to the individual components themselves. The most important goal of the integration testing is to test that components of the system also work together. System tests come next. They are used to observe the whole system, and the focus is on discovering errors in the system level. The testers doing the system testing need to be independent of the development as much as possible to provide credibility and impartiality to the test results (Haikala & Mikkonen, 2011, p. 208). In addition, all non-functional requirements

such as performance testing, security testing, installation testing, and usability testing are included in the system tests. The highest level of testing is the release/acceptance testing. There it is checked that the system meets its requirements and is good enough for external use.

Before the final release of the software, the last version to be tested is called a release candidate. It is a version or a build of the product that can potentially be released to production. A release candidate may undergo even further testing to those mentioned in the previous paragraph or be augmented with documentation and other materials.

The testing itself can be either static or dynamic. They describe what kind of testing is done to the target system, and how it is done. Static testing means that the software is tested without executing any code. This can be by doing a review, a walkthrough, an inspection, or analysis, for example, based on different metrics. It mainly involves syntax checking of the code and manually reviewing algorithms to find faults. Dynamic testing is the opposite of static testing. Dynamic testing means that the code is run, and the responses of the software to given inputs are inspected.

Regression testing is a universal term used to describe the testing done to ensure that earlier discovered errors that have been fixed do not show up again. It is especially used to make sure that the new modifications have not broken anything or created any new errors. (Kasurinen, 2013, pp. 68-69)

In general, regression tests verify that there has not been a change in the behavior of the software. They usually consist of unit tests to check coding or acceptance tests to define the desired system behavior. Those tests that have passed are included in a regression test suite that is meant to guard against new unintended changes. To receive continual feedback from the regression tests they should be automated as much as possible. (Crispin & Gregory, 2009, p. 496)

There are several different tools that can provide help during a testing process. One example is the bug backlog that is used to report all the discovered faults for fixing. The bug backlog can also be used to explore what type of faults the software has had and where those have located. Another helpful tool is a one used to manage the test cases. Test case management helps in keeping the different test cases in order and makes it easier to control adding new test cases and possibly removing obsolete ones. Test management can also be used to select appropriate test cases for each testing level and follow their progress and results. It allows testers easily to examine how many tests have been completed, and how much of them have passed and failed.

The testing process can be assessed and tracked with several different methods and measures. In principle, any retrievable information regarding the status of the system or the project can be used as a measure for testing. In practice, however, there are four re-

quirements for a measure. It needs to replicable, exact, comparable, and economical. Examples of such measures are the number of passing/failing tests, code coverage, or a number of reported/open defects. (Kasurinen, 2013, pp. 162-165)

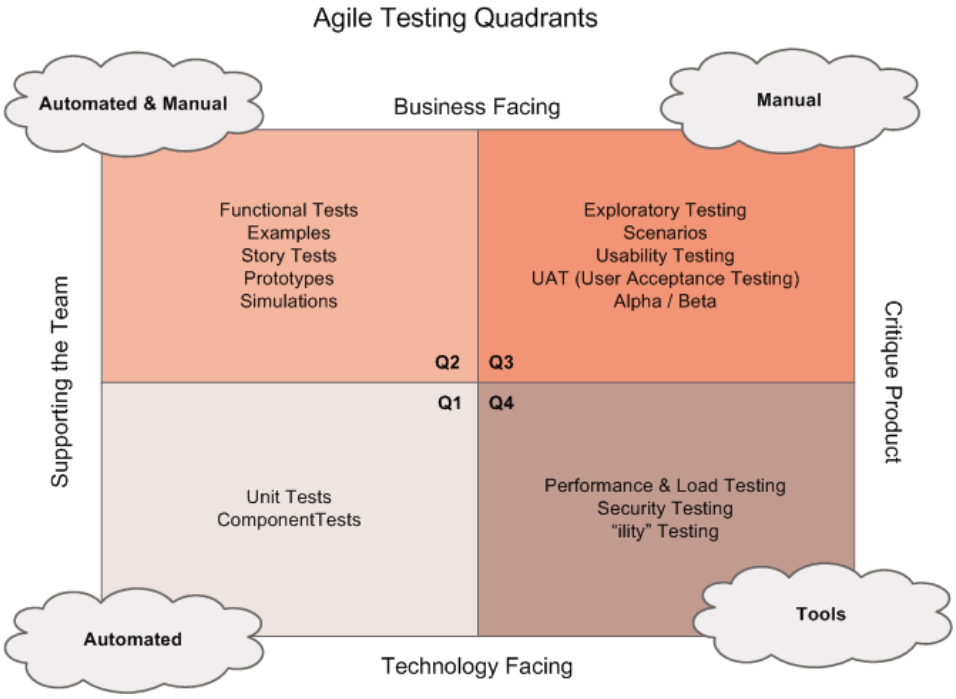

Figure 4. *Agile Testing Quadrants (Crispin & Gregory, 2009).*

In addition to categorizing tests into different levels, they are also categorized based on their reasons. In that approach, one way to determine the different categories is demonstrated in the diagram in Figure 4. This approach is a possibility, particularly in the agile software development process. The diagram in Figure 4 presents the agile testing quadrants that reflect the different reasons to test. The testing reasons are listed as the axis values so that the horizontal axis divides the matrix into tests that support the team and the tests that critique the product. The vertical axis divides them into business-facing and technology-facing tests. (Crispin & Gregory, 2009, pp. 97-98)

2.1.3 Test automation

Automation of software testing makes it possible to conduct more tests in a shorter time and with better coverage than by manual testing. Automation is especially useful and relevant when conducting regression and release tests (Haikala & Mikkonen, 2011, p. 213). Furthermore, automated tests can be effortlessly run even after every modification to ensure that the basic functionality of the software still works. Regardless, manual tests are also relevant and still very much needed because they are more efficient in finding new faults.

Test automation is a core agile practice. Agile software projects depend on automation to be effective. Well-implemented automation frees the development team to deliver high-quality code frequently. It also provides a framework that lets the team maximize its pace while maintaining a high standard. Source code control, automated builds and test suites, deployment, monitoring, and a variety of scripts and tools help eliminate monotony, ensure reliability and allow the team to perform at its peak level continuously. (Crispin & Gregory, 2009, p. 255)

In all, test automation has several benefits over manual testing: (Crispin & Gregory, 2009, p. 258):

- Manual testing is time-consuming.
- Manual processes are error-prone.
- Automation frees people to concentrate on the more relevant work.
- Automated regression tests provide a continuous safety net.
- Automated tests give feedback early and often.
- Tests and examples that drive coding can do more.
- Tests provide documentation.
- Automation can be a good return on investment.

With sufficient coverage, an automated test can easily and more effectively tell far-reaching effects that are very challenging for manual testers to find (Crispin & Gregory, 2009, p. 262). Additionally, automated regression tests prevent manual regression tests for growing in scope and eventually becoming possibly even ignored (Crispin & Gregory, 2009, p. 271).

Moreover, usability testing and explorative testing may also benefit from some automation. This is true especially when setting up test scenarios and analyzing results. Anyhow, it is still important to note that there are things like human instincts, critical thinking, and observation that cannot be automated (Crispin & Gregory, 2009, p. 325).

One of the first things to do when starting to develop test automation is to build a test infrastructure. It may include setting up continuous integration, a test environment, and a test database. Also, an automated build process is needed to provide test results from each build.

Test automation allows running the same tests easily in different environments. This is particularly useful since tests should be done for example in different browsers and operating systems to reveal issues that are related to each specific environment. Manually running those same tests on multiple environments would require too much time, but with test automation test executions can be run in parallel in different environments.

After the testing infrastructure including the testing environment and automated tests have been built, they need to be maintained and updated continuously. Likewise, the test results need to be monitored in order to notice if a test fails and to track down the reasons

for it. If the failure is related to the testing environment, the environment must be fixed. Likewise, if the failure is related to the functionality of the program, it needs to be resolved whether there is a fault in the program or the functionality has changed so that the test has become out-of-date. In case the problem was in the test environment or in case the test case turned out to be out-of-date, the person responsible for test automation needs to update the test case or solve the environment problem. On the other hand, if a fault was found, a bug report should be written and directed to the person responsible for the fault or otherwise able to fix it.

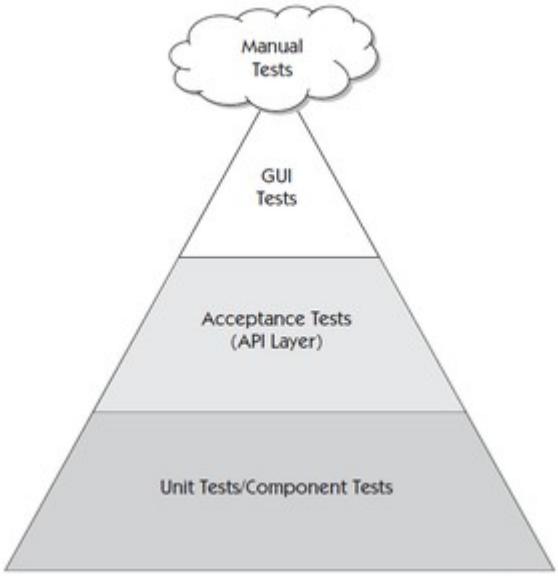

Figure 5. *Test automation pyramid (Crispin & Gregory, 2009, p. 277).*

Figure 5 illustrates the "test automation pyramid" that was introduced by Mike Cohn. There are three different layers of the automated tests shown in the pyramid. The lowest tier is meant to be the foundation that will support all the other tests on top. It is mainly composed of robust unit tests and component tests, in other words, technology-facing tests that support the team. This layer includes most of the automated tests. They are generally written in the same language as the system under the test. The goal is to include as many tests as possible in this layer. (Crispin & Gregory, 2009, p. 276)

The middle tier in the pyramid includes "story" tests, "acceptance" tests, and tests that cover larger sets of functionalities than the unit test layer. These tests operate at the API level or behind the graphical user interface (GUI), in other words, the tests for testing the functionality directly without going through the GUI. Test cases for the middle layer tests include setting up inputs and fixtures that feed the inputs into the production code, accept the outputs, and compares them with the expected results. Since these tests bypass the presentation layer, they are less expensive to write and maintain than tests that use the actual user interface. They do not provide feedback as quickly as the unit level tests because each test covers more ground and may access the database or other components.

Regardless, they are still much faster than doing the same tests through the user interface. (Crispin & Gregory, 2009, p. 277)

The top tier represents tests that are done through the GUI. These tests use the presentation layer by operating and manipulating it. They are written once the code is completed and are usually meant to give critique to the product and, therefore, are directly included in the regression suite. These tests are traditionally more expensive to write although there are new tools that help reduce the investment needed. The components of the user interface tend to be changed often, which makes these tests more sensitive than tests that work at the lower levels. Running tests through the user interface also slows the tests down compared with the tests at the lower levels. Still, the tests in the top layer provide important feedback even though a suite of GUI tests may take hours to run rather than a few minutes required for unit-level test suites. A number of tests on this layer should be minimized so they should only form the tip of the pyramid. (Crispin & Gregory, 2009, p. 278)

Although test automation brings many advantages, most systems also need manual testing activities such as exploratory testing and user acceptance testing. Those should not be forgotten which is why those tests are illustrated with the little cloud at the tip of the pyramid. (Crispin & Gregory, 2009, p. 278)

2.2 DevOps

DevOps (a clipped compound of "development", and "operations") is software engineering culture and practice that aims at unifying software development (Dev) and software operation (Ops). It aims to build an environment where developing, testing, and releasing software can happen fast, frequently, and reliably. DevOps activities rely significantly on the practices and processes of agile software development as well as on the automation tools for infrastructure management.

Another way to explain DevOps is to describe it as an approach based on lean and agile principles in which business owners and the development, operations, and quality assurance departments collaborate to deliver software in a continuous manner. This then enables the business to seize market opportunities more quickly and reduces the time to obtain and process customer feedback (Sharma & Coyne, 2017, p. 1).

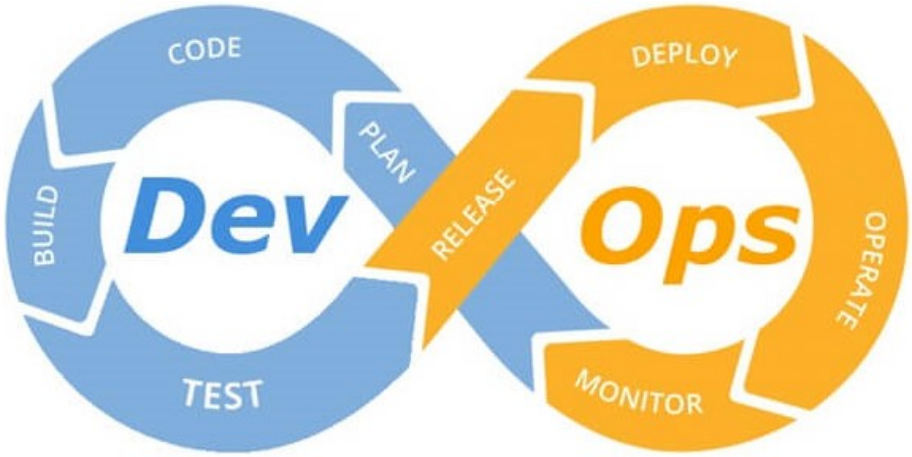

Figure 6. *Stages of a DevOps toolchain (Thomas, 2018).*

An example illustration of the stages of DevOps in presented in Figure 6. It is clearly shown how all the different stages are meant to form a continuous flow where development and operations are closely tied together.

The DevOps movement has produced several principles that have evolved over time and are still evolving. Several variants of the principles have been developed by many solution providers. Nevertheless, all these principles take a holistic approach to DevOps, which makes it possible for organizations of all sizes to adopt them as well. These principles are (Sharma & Coyne, 2017, p. 6)

- Develop and test against production-like systems.
- Deploy with repeatable and reliable processes.
- Monitor and validate operational quality.
- Amplify feedback loops.

The first principle of developing and testing against production-like systems relates to the DevOps concept of shifting left where operational concerns are moved earlier in the software delivery lifecycle so more towards development. The goal is to allow development and quality assurance teams to develop and test against systems that behave like the actual production systems to see how the application behaves and performs well before it has reached the deployment stage. (Sharma & Coyne, 2017, p. 6)

Deploying with repeatable and reliable processes allows development and operations teams to support an agile (or at least iterative) software development process all the way through to deployment. Here, automation plays an essential role when creating processes that are iterative, frequent, repeatable, and reliable. This follows from the goal that the organization must create a delivery pipeline that allows for continuous and automated deployment and testing. Furthermore, frequent deployments allow teams to test the deployment processes themselves which help lower the risk of deployment failures at release time. (Sharma & Coyne, 2017, p. 7)

The next principle of monitoring and validating operational quality moves also monitoring earlier in the lifecycle by requiring that automated testing is done early and often. Monitoring includes both the functional and nonfunctional characteristics of the application. One aspect of monitoring is to record the quality of different metrics and provide results from their analysis. This is particularly needed whenever an application is deployed and tested. Frequent monitoring has also the advantage of providing early warning about operational and quality issues that might occur in production. (Sharma & Coyne, 2017, p. 8)

One goal of DevOps is to enable organizations to react and make changes more rapidly. In software delivery, this goal requires an organization to get quick feedback and then learn rapidly from every action it takes. The principle of amplifying feedback loops calls for organizations to create communication channels that allow all stakeholders to access and act on feedback. (Sharma & Coyne, 2017, p. 8)

Efficient feedback loops also create additional benefits for different sectors of the company.

- Development may act by adjusting its project plans or priorities.
- Production may act by enhancing the production environments.
- Business may act by modifying its release plans.

The most important practices for supporting DevOps in software development are continuous integration and continuous delivery. Continuous integration means including every new functionality as part of the software as soon as it is completed. This way, the current version of the software includes all the functionalities that have been developed so far. Additionally, this enables the deployment of the product at any point in time. Continuous delivery, on the other hand, includes all actions needed to take the software to production. The goal of continuous delivery is to speed up and automate the process of taking the software to production as much as possible.

Continuous integration has become popular with the agile movement. From the developers' perspective, the idea is to regularly integrate their work with that of the rest of the developers on their team and then to test the integrated work. In the case of more complex systems made up of multiple systems or services, the concept means integrating their work with other systems and services as well. The benefit of regular integration of results is to be able to discover and expose integration risks as early as possible. In complex systems, it also exposes both technical and schedule-related as well as known and unknown risks. (Sharma & Coyne, 2017, p. 12)

In all, continuous integration has several goals such as (Sharma & Coyne, 2017, p. 13)

- Enable the ongoing testing and verification of the code.

- Validate that the produced code and other developers' code integrated with it as well as the other components of the application functions and performs as designed.
- Continuously test the application being developed.

Continuous testing means testing earlier and continuously across the development lifecycle which results in reduced costs, shortened testing cycles, and continuous feedback on quality. This process is also known as shift-left testing which emphasizes integrating development and testing activities to ensure building quality in the software as early in the lifecycle as possible. Automated testing and service visualization are needed to implement continuous testing. Service virtualization is used to simulate production-like environments and, subsequently, make continuous testing feasible. (Sharma & Coyne, 2017, p. 13)

As mentioned earlier, monitoring of test results plays an important role in DevOps. The basic principle behind following the outcomes of tests is that after a test passes, it should not fail unless the requirements were changed. If that happens, the test should be updated before the code is modified. Whenever a test fails in a continuous integration and build process, the highest priority is to get the build passing again. (Crispin & Gregory, 2009, p. 179)

Running suite of tests automatically every time new code is checked in helps in discovering regression bugs quickly. This way the change is most likely still fresh in some programmer's mind so troubleshooting it would go significantly faster than if the fault was found weeks later in a specific testing phase. Quickly discovered fault or errors are cheaper to fix. (Crispin & Gregory, 2009, p. 262)

The principles of continuous release and deployment take also the concept of continuous integration to the next level. The same approach that enables release and deployment also enables the creation of a delivery pipeline. This pipeline facilitates the continuous deployment of a software product to the quality assurance phase and then to production in an efficient and automated manner. The aim of continuous release and deployment is to release new features to customers and users as soon as possible. (Sharma & Coyne, 2017, p. 13)

2.3 Artificial intelligence (AI)

2.3.1 Definition of AI

Artificial Intelligence (AI) is the science of making things smart. It can be defined for example as "Human intelligence exhibited by machines, especially computer systems". AI applications rely on several technologies that include speech recognition, machine learning and planning, problem-solving, robotics, and cognition for initial observation of

unique events and situations. AI can be divided into different branches that are illustrated in Figure 7. AI itself is a branch of Computer Science.

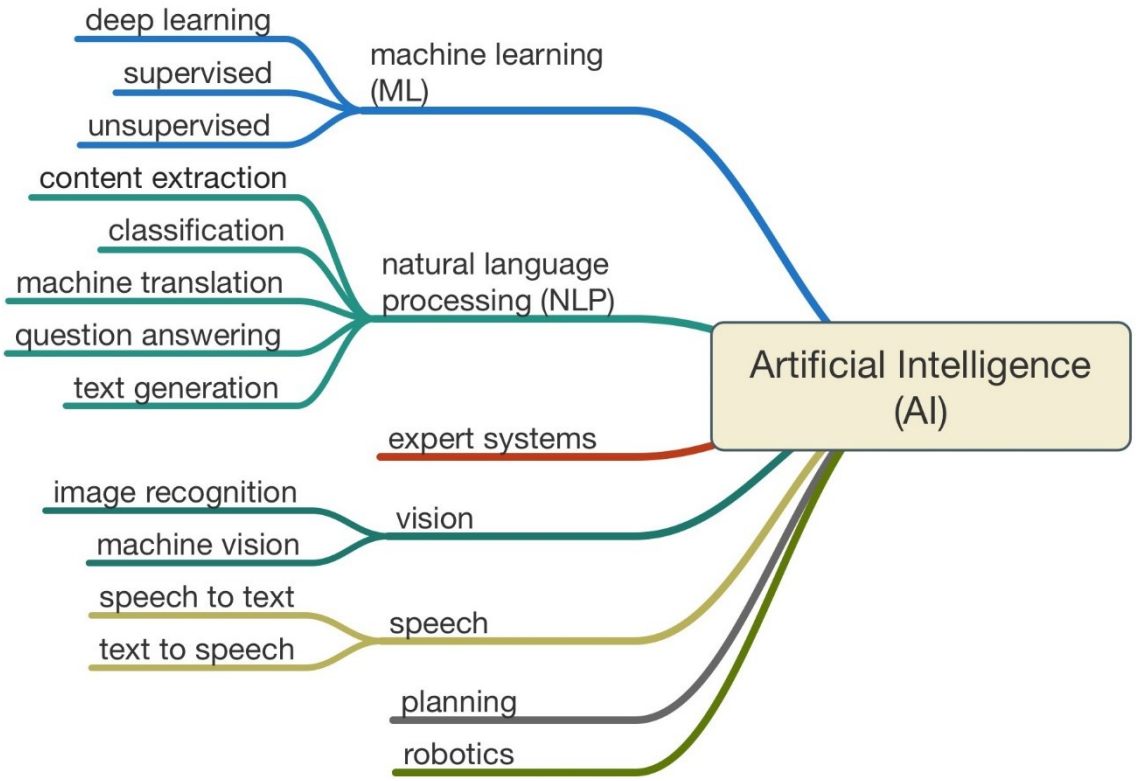

Figure 7. *Branches of Artificial Intelligence (Mills, 2016).*

Machine learning

Machine learning is a form of AI that enables a system to learn from data rather than through explicit programming. It can be defined more specifically as an approach to achieve intelligence through systems that can learn from experience to find patterns in a set of data. In other words, machine learning involves teaching a computer to recognize patterns by example rather than programming it with specific rules. Those patterns can be found within data.

In practice, machine learning is about creating algorithms (or a set of rules) that learn complex functions (or patterns) from data and make predictions on it. This happens by feeding an algorithm a significant amount of input data. The amount of required data depends on the type and structure of the data as well as the type of the problem or a task at hand. The machine learning algorithm then uses this data to learn patterns in it and to form a model based on those patterns. This model is stored and used to make predictions on the new and previously unseen input data.

Machine learning is embodied by different learning approaches which are themselves implemented with various frameworks. Examples of some of those learning paradigms

include supervised, unsupervised, semi-supervised, regression, and reinforcement learning. The difference between the different approaches is generally in the availability of data labels. In supervised learning, the data labels are available and generally discrete. In unsupervised learning, the data labels are unavailable. Semi-supervised learning lies in between so that there are some data labels available but not all. Those labels are generally discrete. Regression learning, then again, uses continuous data labels. The last paradigm, reinforcement learning, is based on an agent policy optimization in a rewarding setting. (Japkowicz & Shah, 2011, p. 1)

A more specific definition of reinforcement learning is learning by trial-and-error through reward or punishment. For example, let us have a program learn by playing a game millions of times. While playing, the program is rewarded when it makes a good move. This strengthens the connection to make moves like that. When it loses, no reward (or negative reward) is given. Over time, the program learns to maximize the reward without a person explicitly telling the rules. This approach may lead to a better performance than could be achieved by a human because the program can find plays that no one ever thought of doing before.

It is possible to use machine learning for all kinds of tasks. These tasks can be categorized based on, for example, the desired outcome of the task. Machine learning tasks can be

- Classification where the input data is split into two or more classes and the goal is to produce a model that can assign new unseen inputs to one or more (multi-label classification) of these classes. A supervised learning is usually used for classification tasks. An example of a classification task is spam filtering where the input data consists of email (or other) messages and the classes are "spam", and "not spam".
- Regression which is also considered a supervised problem with the difference that the outputs are continuous rather than discrete.
- Clustering which means grouping a set of inputs into different output groups. These groups are not known beforehand and are determined by the algorithm based on the structure of the input data which makes this typically an unsupervised task.
- Density estimation which is used to find the distribution of input data in some space.
- Dimensionality reduction where the input data is simplified by mapping it into a lower-dimensional space. An example of such a task is topic modeling where a program is given a list of human language documents and is tasked with finding out which documents cover similar topics.

When facing any of the tasks mentioned above, a machine learning algorithm needs to be chosen. There are multiple alternatives for that. For example, the K-nearest neighbors

algorithm, Linear regression, Naive Bayes classifier, Linear classifier, Support vector machines, Random Forests, and Artificial neural network. In machine learning, artificial neural networks (loosely modeled on the brain) are used to calculate probabilities for features they are trained to look for. It should be noted that most of these algorithms are not optimal or even usable in all tasks. For example, the K-nearest neighbor algorithm is best suited for clustering tasks.

It is most advantageous to use machine learning in those computing tasks where designing and programming explicit algorithms with good performance are difficult or infeasible. Examples of such applications include email filtering, anomaly or fault detection, the detection of network intruders or malicious insiders working towards a data breach, optical character recognition (OCR), learning to rank, and computer vision.

An additional example of a machine learning task is text classification. It is also known as categorization where input data is some text, and the task is to decide which of the predefined set of classes it belongs to. Text classification includes for instance language identification and genre classification as well as semantic analysis (classifying a movie or product review as positive or negative) and spam detection (classifying an email message as spam or not-spam). (Russell & Peter, 2010, p. 865)

Natural language processing (NLP)

Natural Language Processing (NLP) is defined as the ability to train computers to understand both written text and human speech. Its techniques are needed to capture the meaning of the unstructured text from documents or communication from the user. That makes NLP the primary way for systems to interpret text and spoken language. NLP is also considered one of the fundamental technologies to allow non-technical people to interact with advanced technologies. For example, NLP can help users ask a system questions about complex data sets without the need for coding. Structured database information relies on schemas to add context and meaning to the data whereas unstructured information must be parsed and tagged to find the meaning of the text. This parsing and tagging can be tackled by NLP. Implementation of NLP requires different tools such as categorization, ontologies, tapping, catalogs, dictionaries, and language models. (Hurwitz & Kirsch, 2018, p. 14)

Information retrieval is a task where the aim is to select a subset relevant to a query from a collection of textual documents. The selection is made based on a keyword search and is possibly augmented using a thesaurus. An information retrieval process usually returns a ranked list of documents where the rank is a system assigned measure of relevancy to the document in response to the query. This ranked document list, however, does not provide any detailed information on the content of the documents. (Poibeau, Saggion, Piskorski, Yangarber, & (eds), 2013, p. 25)

Information extraction, on the other hand, is the task of automatically extracting structured information from unstructured and/or semi-structured machine-readable documents. In most cases, this activity makes use of NLP in processing human language texts. Recent activities in multimedia document processing like automatic annotation and content extraction out of images, audio, or video could be considered as information extraction. (Poibeau, Saggion, Piskorski, Yangarber, & (eds), 2013, p. 24)

Compared with information retrieval, the goal of an information extraction process is not to rank or select documents, but to extract the main facts from the documents. These facts should be about pre-specified types of events, entities, or relationships and they are then used to build more meaningful and rich representations of their semantic content. The facts could again be used to populate databases that provide structured input to mining more complex patterns (e.g., trends, summaries) in text collections. To summarize, information extraction aims to process textual data collections into a shape that facilitates searching and discovering knowledge in such collections. (Poibeau, Saggion, Piskorski, Yangarber, & (eds), 2013, p. 25)

More generally, information extraction proceeds to identify a predefined set of concepts in a specific domain ignoring other irrelevant information. A domain, in this case, consists of a corpus of texts together with a clearly specified information need. In other words, information extraction is deriving structured from unstructured text. An example of an information extraction task is extracting information on violent events from the online news with the interest in identifying the main actors of the event, its location, and the number of people affected. (Poibeau, Saggion, Piskorski, Yangarber, & (eds), 2013, p. 24)

One of the subtasks of information extraction is named entity recognition (NER). Its aim is to find each mention of a named entity in the text and label its type. What constitutes a named entity type is application specific and can typically include people, places, and organizations. More specifically, anything that can be referred to with a proper name can be considered as a named entity. Furthermore, the term is commonly extended to include other things like dates, times, and other kinds of temporal expressions as well as numerical expressions like prices. (Jurafsky & Martin, 2017, pp. 348-349)

Named entity recognition consists of finding spans of text that constitute as proper names, and then classifying the names to their respective entity. This is a challenging task partly because of the ambiguity of segmentation. It needs to be defined what is an entity and what is not and where the boundaries are. Another challenge is caused by type ambiguity. For example, the abbreviation JFK can refer to a person, the airport in New York, or any number of schools, bridges, and streets around the United States. (Jurafsky & Martin, 2017, p. 350)

One tool used in NLP implementations are regular expressions that are considered as the standard notations for characterizing text sequences. Formally, a regular expression is an algebraic notation for characterizing a set of strings. Thus, they can be used to specify search strings as well as to define a language in a formal way. Regular expressions are used for specifying text strings in situations like web searches and in other information retrieval applications. Moreover, they also play an important role in word-processing, computation of frequencies from text collections, and other such tasks. (Jurafsky & Martin, 2017, pp. 21-22)

2.3.2 AI development process

When developing software that uses artificial intelligence, the development process can be divided into phases. First, the problem needs to be identified. Next, an adequate approach to solve the problem must be found and enough applicable data for the development process and testing needs to be gathered. Then, the solution/model is developed/programmed. If a machine learning approach is used, the next phase is to train the model with the data. After training, the model is optimized by trying out different parameter values and by fine-tuning its other features by repeating the previous phase. Finally, the optimized model is tested with a separated testing set.

Machine learning development process

It is typical that a generated machine learning model uses the training data to develop itself so that it would be able to produce a better solution to the problem. The problem could be any of the machine learning tasks mentioned in the previous section. An example could be a classification task where the material is to be classified to a correct category.

One of the most central products in a machine learning development process is the model. It is the output generated when you train your machine learning algorithm with data. After training, the model can be used to produce an output for a given input data. The type of model will depend on the type of algorithm used for its generation. For example, a predictive algorithm will create a predictive model. When provided with data, the model will produce a prediction based on the data that was used to train it. Nowadays, machine learning has become an essential component in creating analytics models. (Hurwitz & Kirsch, 2018, p. 4)

Machine learning algorithms differ from other algorithms. For most algorithms, a programmer starts by formulating the algorithm. However, in the case of machine learning algorithms, the process is flipped. This means that the data itself creates the model. The more data added to the algorithm, the more sophisticated the algorithm becomes. It is common that the accuracy of a machine learning model improves when it is exposed with more and more data. (Hurwitz & Kirsch, 2018, pp. 28-29)

Data and its processing have an important role in machine learning development. In particular, the quality of data plays an essential role. In order to build a model that generalizes well also to data that is outside of the training data, the training data needs to contain enough information that is relevant to the problem at hand. To make sure that data is in the most optimal format for the machine learning algorithm, it needs to be preprocessed. Data preprocessing includes cleaning, instance selection, normalization, transformation, feature extraction, and selection, etc. If you create a model based on faulty data, your predictions will obviously be inaccurate.

There are three different data sets that are needed in the development process: training, validation, and testing sets. Each of the data sets is used in the corresponding phase. It is essential for the accuracy of the resulting model that the data sets are kept separate from each other. The algorithm is inductively built by observing data from training and validation sets. The data from the training set is used to teach the algorithm. Then, the data from the validation set is used to optimize the parameter values. The test set data is used for testing the accuracy of the algorithm. Accuracy is measured by comparing the output of the algorithm with the right answers according to the test set.

One of the common risks in developing machine learning models is overfitting. Overfitting means that the model is precisely tuned for the training data but may not be applicable to the large sets of unknown data. To avoid overfitting, enough data is needed. Another protective measure against overfitting is to do testing against unforeseen or unknown labeled data. Using unforeseen data for the test set can help evaluate the accuracy of the model in predicting outcomes and results. (Hurwitz & Kirsch, 2018, p. 15)

Another common risk is caused by biased data. Once it is determined which features should be used, the biggest challenge is finding enough unbiased training data for all of those features in a format that can be fed into the machine learning system for training. It is important to understand that a machine learning system cannot predict things it does not know about. For example, if a system that is trained to recognize a dog and a chicken from the image were given an image of a cow, it would probably predict that there is a dog in the image.

The machine learning cycle is the process used when developing a machine learning model. It is continuous and choosing the best machine learning algorithm is just one of the steps. The steps in the cycle can be listed as follows: (Hurwitz & Kirsch, 2018, p. 37)

- Identify the data: The first step of the cycle is to identify the relevant data sources. In addition, while the development process proceeds, the target data set should be expanded to improve the system.
- Prepare data: The data should be cleaned, secured, and governed. If a machine learning application is created based on inaccurate data, the application will fail.

- Select the machine learning algorithm: There are several algorithms available, and there can easily be more than one that is applicable to the specific task at hand.
- Train: The algorithm needs to be trained to create a model. Depending on the type of data and the algorithm, the training process may be supervised, unsupervised, or reinforcement learning.
- Evaluate: The model needs to be evaluated to find the best performing algorithm.
- Deploy: Machine learning algorithms create models that can be deployed to both cloud and on-premises applications.
- Predict: After deployment, the model is used to make predictions based on new incoming data.
- Assess predictions: The validity of the obtained predictions is assessed. The information gathered from analyzing the validity of predictions is then fed back into the machine learning cycle to help improve the accuracy of the model.

NLP/NER development process

Traditionally, work in natural language processing has tended to split the process of language analysis into several stages according to the theoretical linguistic distinctions drawn between syntax, semantics, and pragmatics. The more practical explanation is that the sentences of a text are first analyzed in terms of their syntax. This provides an order and structure that is more amenable to an analysis in terms of semantics. Next, a stage of pragmatic analysis is conducted whereby the meaning of the utterance or text in context is determined. These stages along with a couple more are illustrated as separate boxes in Figure 8. However, it is recognized that in real terms it is not so simple to separate the processing of language into neat boxes. Regardless, such an illustration can be used as a pedagogic aid and could also serve as the basis for architectural models that make the task of natural language analysis more manageable from a software engineering point of view. (Indurkhya & Damerau, 2010, p. 4)

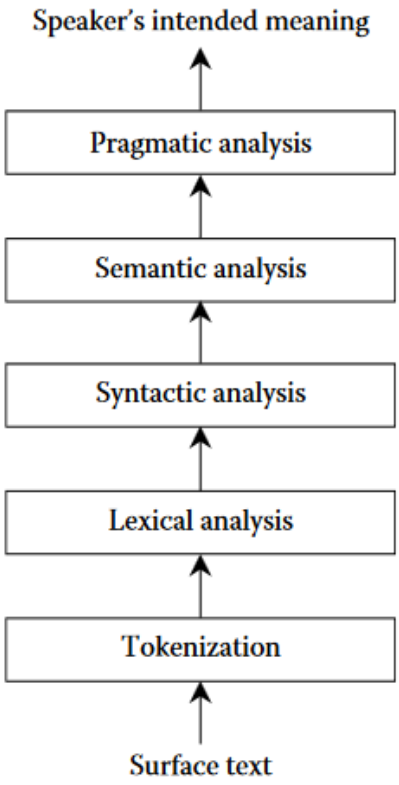

Figure 8. *The stages of analysis in processing natural language (Indurkhya & Damerau, 2010, p. 4).*

Instead of relying on the traditional analysis process, the development of a named entity recognition (NER) system could be based on a similar cycle as used for developing a machine learning system. It starts by defining the problem i.e. determining the expressions that need to be recognized from the text. Examples of such expressions are person's name, company's name, and a price of a product. Next, the actual algorithm to solve the problem is planned and implemented. The implementation is optimized once it is seen how it performs. As a final phase, the algorithm is tested with realistic data. This test data is most likely text with annotations so that the target expressions are marked. The marked words are then compared with the ones produced by the algorithm.

Data does not necessarily play such a significant role in developing NER systems as it did for machine learning systems. Although, it is possible to implement an NER system by using machine learning based statistical modeling where the case data has again an essential role in training the model, and a large amount of manually annotated training data is typically required. However, there are also other ways to implement an NER system. One option is to use regular expressions that can be used in text and pattern matching. Regular expressions can then be used to pick up numbers, words, or phrases with a specific format from the text. Another option would be to use lists whose content is searched and picked up from the text. It has been found that hand-crafted grammar-based systems typically obtain better precision, but it does so at the cost of a lower recall and months of work by experienced computational linguists (Kapetanios, Tatar, & Sacarea, 2014, p. 298).

2.3.3 Testing AI systems

Artificial Intelligence (AI) and software testing can be combined in several ways. For example, it can be so that the tested system includes AI or that AI is used as a tool to help software testing. An option for the latter case is to use a machine learning algorithm to estimate the testing time (Cheatham, Yoo, & Wahl, 1995). Additionally, machine learning can be used to generate test data.

In this thesis, the focus is on testing software that uses AI. The AI has already been implemented and optimized. Hence, the aim of this thesis is to ensure that the testing software does not malfunction or break when the AI component is integrated with the other testing software components.

However, before going into the specific implementation of the testing software, testing of AI systems is introduced more generally. Especially the differences between AI related testing and regular testing are covered.

Why the testing of AI systems is so difficult?

There are two phases and two datasets in testing AI. Those phases are validation and testing. The validation set is used to evaluate a given model, and it used for frequent evaluation. This data is used to fine-tune the model hyperparameters. Hence, the model occasionally sees this data, but it never actually learns from it. In all, the validation set in a way affects the model, but it does so indirectly. The test dataset, on the other hand, provides the golden standard used to evaluate the model. It is only used once the model is completely trained.

Almost all the research on software testing focuses on the development and analysis of the input data. There is an underlying assumption that once this phase is complete the remaining tasks are straightforward. These consist of running the program on the selected data and producing an output that is examined to determine how well the program performed on the test data. (Weyuker, 1982)

The mechanism for checking the performance accuracy is known as an oracle. The oracle's assumption is that the tester is routinely able to determine whether the test output is correct. Intuitively, it seems sensible to require that a tester can determine the correct answer in a reasonable amount of time while using a reasonable amount of effort. Therefore, if it happens that there does not exist an oracle or it is practically too difficult to determine the correct output, the program should be considered as non-testable. (Weyuker, 1982)

Some programs fall into the category of non-testable programs. Such programs include programs that were written to determine the answer. This is understandable since if the correct answers were known, there would not have been the need to write the program in

the first place. Other examples of non-testable programs are those that produce so much output that it is impractical to verify all of it and those for which the tester has a misconception. In the latter a case, there are usually two distinct specifications, and the tester is comparing the output against a specification which differs from the original problem specification. (Weyuker, 1982)

Machine learning applications can turn out to be non-testable programs that means that there is no reliable test oracle to indicate what the correct output should be for arbitrary input. Machine learning applications mostly fall into the first category of programs that were written to determine the answer.

When there is not an unambiguous answer to the behavior of a program, problems will arise when considering the automation of the testing process. Automation may know how to interact with the system, but it is unable to distinguish between the correct and incorrect behavior of the program. Here, the definition of correct behavior comes from the source of truth that can be, for example, a product owner, a stakeholder, or a customer.

Examples of non-testable machine learning programs are creating a summary or picking up keywords from a text. In these cases, a person could define what is correct and what is not. However, when asking from several people, each one will probably give a different answer. In these cases, the answer is mostly a matter of opinion.

Nevertheless, not all machine learning programs are non-testable, and it is possible to define the right answers to them. Still, it is often necessary for a person to define what is correct. For example, when classifying images of cats and dogs, the right answer can be defined, but a person is needed to mark images accordingly before they could be used for training and testing.

One challenging example is an NER program that recognizes human names from the text. In principle, a person can search and mark the real names from the text, but it is not always that straightforward. For example, April could be either a month or a woman's name. In these cases, the correct meaning can be deducted from the context.

A part of testing is selecting an evaluation metric to evaluate the problem at hand. Selected evaluation metric can affect significantly the results of determining whether the solution is good enough for a specific purpose. Different evaluation metrics are covered in more detail later.

Proper testing also needs a large-enough test dataset. The requirements for the test dataset are similar to those of a machine learning training dataset. It needs to be unbiased and preprocessed. There need to be enough data samples for the selected evaluation metric to give good results.

Test dataset

In addition to the previously mentioned requirements, the test data needs to be realistic which means that it should resemble as accurately as possible the data of the end user. Furthermore, it is possible that the final product is used for different languages. Hence, the test data is needed from all those languages, and there needs to be enough of it for each language to achieve statistically significant results.

Even though it is important to have a large amount of test data, the amount is limited by the fact that a person needs to go through the data and mark what is correct. In case of a classification task, a class needs to be defined for every image or a file. If it is a named entity recognition task, annotation is needed for the whole data set. Therefore, a person is needed to mark all wanted entities like persons' names from every document. In conclusion, the test data should be large enough to cover a breadth of use cases and at the same time, it must be balanced against the time required for human processing.

Annotation

In some cases, the correctness of the results can only be defined by a person. This is the case, for example, in information extraction and named entity recognition. A person knows what entities are to be extracted. The process of marking interesting things from a text is called annotation. Those markings can then be compared with the entities extracted by the model. In more general terms, annotation is defined as adding metadata to a specific span of text. In the annotation context, an entity is something that has a distinct and separate existence independent of the text. A textual realization of an entity is called a mention. In other words, a mention is how the entity is explicitly presented within the text. The label is the word or phrase that is used to describe an annotation. Its purpose is to tie the annotation and its mention in the text to an entity. (Petrillo & Baycroft, 2010)

Annotations can be added to the data by two principal ways: automatically or manually. Automatic annotation is less precise but can cover many more documents than manual annotation. Manual annotation, on the other hand, is more precise (to a point), but very labor-intensive. It is often used to train a machine to perform automatic annotation. (Petrillo & Baycroft, 2010)

There are some tools for manual annotation. For example, Brat is a free open source web-based tool for text annotation. There is an example image from Brat in Figure 9.

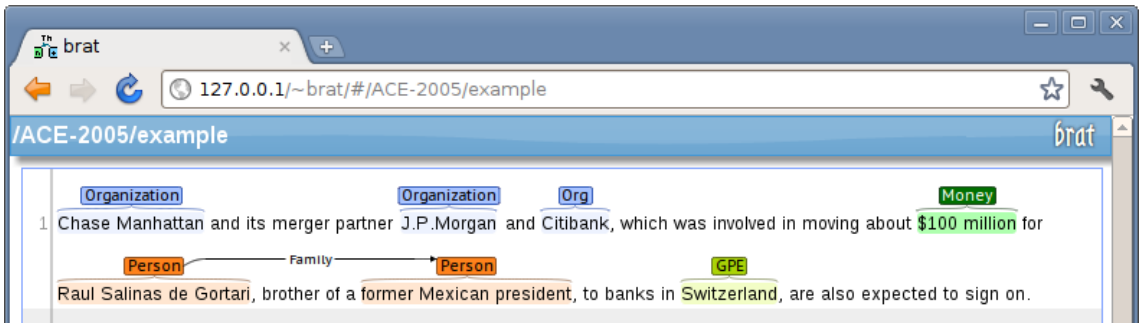

Figure 9. *Brat annotation tool (Brat, 2018).*

Figure 9 shows a simple example where a sentence has been annotated to identify the mentions of some real-world entities (things), their types, and a relation between the two. This example illustrates two basic categories of annotation: (Brat, 2018)

- Text span annotations, such as those marked with the “Organization” and “Person” types in Figure 9
- Relation annotations, such as the “Family” relation in Figure 9.

The text span category is suitable for creating annotations for named entity recognition, and binary relations are best suited for simple relational information extraction tasks. (Brat, 2018)

In the annotation process, specific spans of text are associated with specific labels that are often used later for some sort of processing. In most cases, manual annotation will follow strict guidelines describing what and how to annotate as well as how to label the annotations. There are some general guidelines for annotation. They will almost always be accompanied by precise guidelines specific to the project, including (Petrillo & Baycroft, 2010)

- What to annotate
- Which labels to apply in which circumstances and
- How to address special cases.

Usually, a guide is used to define what should be annotated so that a person knows what to annotate. This guide defines precisely when, for example, a person’s name is annotated and when not. In some cases, it can be that only the first or the last name is annotated. Guides are also needed because people can understand nonspecific instructions differently. Congruence is essential because annotations are used as the right answers when evaluating systems.

Guides are important in tackling different questions and challenges that are likely to arise during the annotation process. However, there will be situations that cannot be covered by explicit rules. For example, if the text includes the color "Navy Blue" and the system only suggested the color "Blue", is the system correct or is it partly correct? If it is partly correct, how is that quantified? Unfortunately, this is an example of a situation without

an explicit rule to cover it. In these cases, the most important guidance is to be as true as possible to the use case that is being modeled. It may help consider what the user would expect in the situation. What is essential though is that whatever decision is made, it must be done consistently throughout the test as well as noted along with the findings.

There is another challenge that often arises with respect to the specification of the problem. It might be that two of the human annotators have a slightly different understanding of the definition of colors. One may be a purist and does not consider black and white to be true colors of the rainbow and skips over them while the other annotator includes them. This sort of inconsistency can often be resolved early with a clear definition or at the very least, during cross-reference validation. It may also be beneficial to allow the human annotators to shape their own definition based on their understanding of the problem. This can also help reveal potential issues with the system definition and/or user expectations of the system.

Similarly, there can be many other subjective, and debatable situations as the following simplistic examples demonstrate. Is the word green, as in environmentally green, referring to a color? What about white, as in the White House? Or blue, as in blue jay? These examples bring out once more the importance of being true to the use case and having consistent annotations.

The manual annotation has its own process. It would be possible to read a document from start to end and mark all annotations in order as they are encountered. This has not proved to produce the most accurate results. Instead, a more methodical approach must be applied to ensure the consistency of the results. The approach should be followed by all annotators, and it includes the following actions in the defined order: (Petrillo & Baycroft, 2010)

1. Reading the whole document. The idea is to read to get an understanding and not to mark any annotations.
2. Marking the entities. The document is read for the second time while marking the entities.
3. Looking again. The markings are reviewed so that nothing has been missed and that the annotation types and features are correct.
4. Recording any additional information. While annotating, all important comments should be recorded. The text file or a Wiki page can be used of this purpose. Examples of recorded comments are:
    a. Whether annotation decision was hard to make
    b. Any question or uncertainty concerning the annotations and guidelines
    c. Anything unclear or ambiguous in the guideline
    d. Things that are considered important and were not covered by the allowed annotations
    e. Annotation tool bugs and/or issues.

There are also techniques to ensure the correctness of the human annotations. Examples of such techniques are cross-validation and spot checking. Those techniques are usually implemented so that multiple people annotate the same text, and then those annotations are merged. This way a single person's opinion does not affect too much. However, a challenge arises when the same text is annotated multiple times, and annotations need to be combined to one result. If there are contradictions between the annotations of different people, the challenge is how to combine those.

Evaluation metrics

When evaluating a system that produces machine-generated output, e.g. predictions or suggestions, it is important to somehow qualify the correctness of the system. This is can be done by comparing the output of the machine with something that has been created or at least validated by one or more people.

When using different evaluation metrics, it is important to understand what each of them promises and interpret their results. Since the tests are not definitive, it is important that their meaning and their results are interpreted properly. (Japkowicz & Shah, 2011, p. 6)

To be able to calculate the actual evaluation metrics, the terms true positive (TP), false positive (FP), true negative (TN), and false negative (FN) need to be defined. True positive describes those elements that are selected correctly by the system. False positives describe elements that are wrongly selected, in other words, elements that were selected even though they should have not been. True negative refers to those elements that are correctly unselected. Lastly, false negative is used to describe the wrongly unselected elements, in other words, those that should have been selected but were not. All the terms are illustrated in Figure 10.

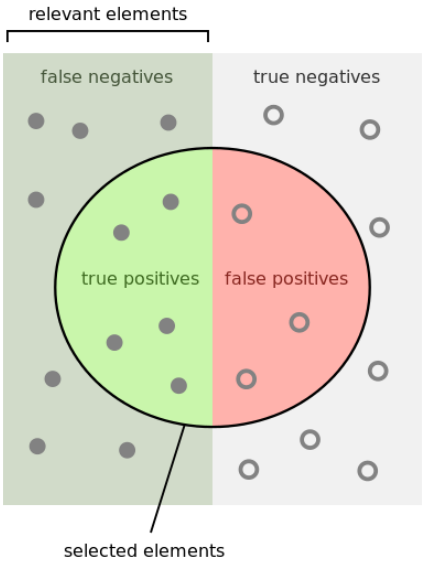

Figure 10. *Definition of true positive, false positive, true negative, and false negative. (Walber, 2014)*

Accuracy measures the goodness of a model as the proportion of true results to the total number of cases. It is not always applicable since it requires counting the number of true negatives which might not be possible or meaningful.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

In cases, where the test data is unbalanced i.e. most of the instances belong to one of the classes, or there is clearly more interest in the performance on either one of the classes, the accuracy does not really capture the effectiveness of a classifier. Then, there are other more useful metrics.

Precision is the proportion of true results of all positive results i.e. how many of the selected items are relevant. (Japkowicz & Shah, 2011, pp. 94-101)

$$Precision = \frac{TP}{TP + FP}$$

Recall (also known as sensitivity) is the fraction of all correct results returned by the model i.e. how many of the relevant items are selected. (Japkowicz & Shah, 2011, pp. 94-101)

$$Recall = \frac{TP}{TP + FN}$$

F-measure is a metric to combine precision and recall. More specifically, the F-measure is a weighted harmonic mean of precision and recall. It has several variations, but the most commonly used is the F1 Score. The F1 Score is the harmonic average of precision

and recall. It is a balanced measure that weights the precision and recall evenly. F1 Score reaches its best value at 1 (perfect precision and recall) and worst at 0. (Japkowicz & Shah, 2011, p. 103)

$$F_1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

AUC measures the area under the curve plotted with true positives on the y-axis and false positives on the x-axis. The curve could be any curve, but usually, it is the Receiver Operating Characteristic (ROC) curve, and it describes the relation between the true positive rate and the false positive rate. The true-positive rate is also known as recall and the false-positive rate as the fall-out or probability of false alarm. The closer this curve is to the upper left corner, the better the classifier's performance. An example of this is demonstrated in Figure 11. This means maximizing the true positive rate while minimizing the false positive rate. AUC as a metric is useful because it provides a single number that lets you compare models of different types. (Tape, n.d.)

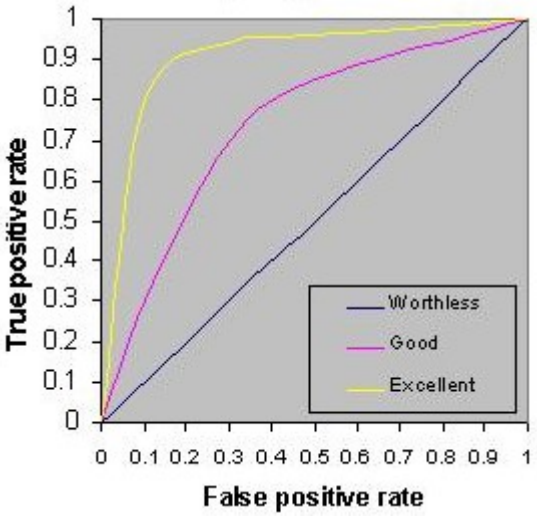

Figure 11. *Comparing ROC curves. (Tape, n.d.)*

Precision-recall Curves, sometimes abbreviated as PR curves, resemble ROC curves in that they explore the trade-off between the positive examples and the number of negative examples. PR curves are formed by plotting precision as the function of recall. In other words, it measures the amount of precision that can be obtained as various degrees of recall are considered. For example, when considering document-retrieval systems, PR curves would plot the percentage of relevant documents identified as relevant against the percentage of relevant documents deemed as such with respect to all the documents in the sample. Unlike the ROC curves, the PR curves have a negative slope and, therefore, look different. This is because precision decreases as recall increases. (Japkowicz & Shah, 2011, p. 132)

2.4 M-Files

M-Files is a Finnish software company that specializes in enterprise information management solutions. Its revenue was 52.6 million in the year 2017 and growth compared with the previous year's revenue of 38.6 million was 37 percent (M-Files Corporation, 2018c). The company's headquarters is located in Hervanta, Tampere, Finland. Other company offices are in the United States, the United Kingdom, France, Germany, Canada, Sweden, and Australia (M-Files Corporation, 2018b).

In August 2018, M-Files Corporation announced the acquisition of Apprento, a Canadian-based provider of artificial intelligence and natural language processing technology solutions. Apprento's technology helps in streamlining the process of classifying, processing, and securing business information by drawing intelligence from the text in unstructured content. In addition, the technology provides contextual insights on related content assets and workflows. The Apprento Business Context Engine uses patent-pending technology for natural language processing (NLP) and natural language understanding capabilities to understand both semantics and concepts in content and communication systems. (M-Files Corporation, 2017)

2.4.1 M-Files as product

The main product of M-Files is an enterprise content management software also called M-Files. Unlike traditional folder-based systems, M-Files organizes objects by their metadata. Metadata consists of information about the object's properties, such as the parties of a contract or the recipient of a letter. The approach is to understand more what the file is and what it represents, and not so much where it is saved.

M-Files is available in multiple languages. In 2018, it supports 32 different languages including Chinese, Finnish, Hebrew, and Spanish.

The M-Files document management software consists of the following components: M-Files Desktop, M-Files Admin, M-Files Desktop Settings, Show Status, and M-Files Server. M-Files can also be used with a web browser or a mobile device. (M-Files Corporation, 2018d)

An M-Files vault is a centralized storage location for documents and other objects. Its physical location is on the server running M-Files Server. Regardless of the physical location, all users see the document vault as a directory on the local M-Files drive of their computer. This means that using a document vault is like using a local hard drive. Vault applications are pieces of software that are installed in the document vault to extend the functionality of the vault. (M-Files Corporation, 2018d)

The Intelligent Metadata Layer (IML) was added to the M-Files 2018 version. It is a repository-neutral approach to intelligent information management that unifies information across the enterprise based on context instead of the system or folder where the information is stored. IML uses artificial intelligence to categorize documents and records automatically and to provide metadata suggestions. (M-Files Corporation, 2018d)

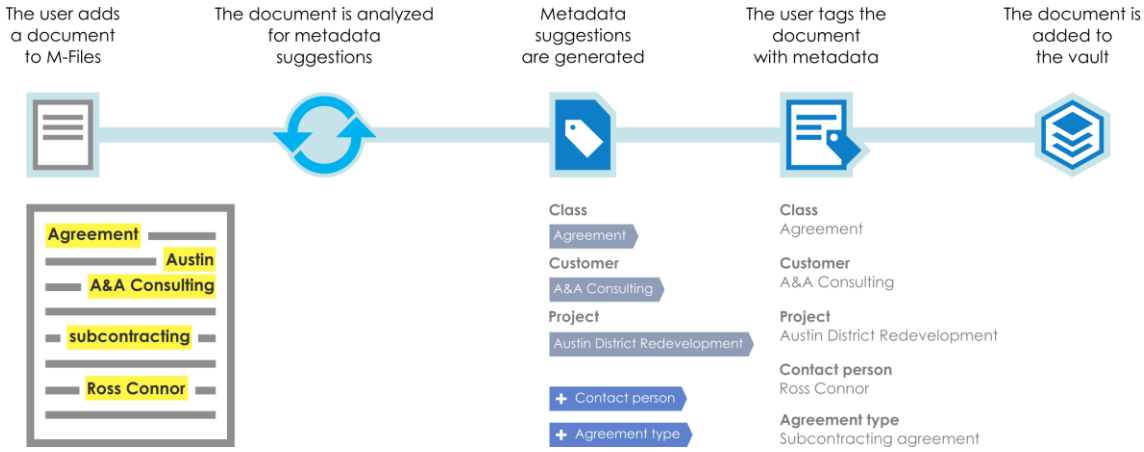

Figure 12. *An example of how intelligence service aids a user in tagging documents with metadata. (M-Files Corporation, 2018d)*

Intelligence services are vault applications designed to analyze and classify documents and offer metadata suggestions based on file contents, existing metadata, and even user behavior. They make use of technologies such as text analytics and machine learning to define and maintain document metadata for the user. An example of how the intelligence service workflow for aiding the user in tagging documents with metadata proceeds is shown in Figure 12. (M-Files Corporation, 2018d)

2.4.2 Testing of M-Files

M-Files is moving toward new principles, Cloud First and DevOps. A Cloud First strategy means that the viability of cloud deployment is always explored first prior to deploying within a more traditional architecture (Ortiz, 2016). DevOps, on the other hand, as explained earlier means aiming at shorter development cycles, increased deployment frequency, and more dependable releases, all in close alignment with business objectives.

Those principles mean that test automation has an important role when assuring the quality of the product. Both the cloud and traditional version of the product need to be working all the time, and that is why testing needs to be done for both versions. There is not the time for long system testing periods because the product release cycle is short, for example, once a month. It also means that the product should be in good, releasable shape all the time with no major issues. To achieve this, additional test automation phases are

needed to get faster test automation feedback to developers. Also, test automation coverage must be increased so that the quality of the product stays good even though there is not the time for long manual testing periods.

Testing process

Testing of M-Files is divided into four phases: testing during software development, test automation, release acceptance testing of M-Files releases, and periodical testing tasks. Issues and improvement ideas that are discovered in any of the phases are documented to M-Files vault that is designed for fault management.

1. Testing During Software Development Phase

The new software versions are developed in M-Files R&D through user stories. Development and testing of user stories are done in biweekly sprint cycles. During sprints, mainly manual testing methods are used to test the user stories. The user stories and their test results are then reviewed and approved according to Definition-of-Done which describes the defined criteria for code reviews, documentation, security and testing. (M-Files Corporation, 2018e)

The testing process starts when a user story is assigned to a tester for testing according to the specified process. It is recommended that the developer arranges a demonstration session for the assigned tester. Also, other members of the Quality Assurance team can be invited to these demonstrations. Next, the test cases are designed. After those have been done, they are reviewed with the developer to ensure sufficient coverage. User story testing is done mainly based on black box techniques, and the test cases are designed so that different use cases, configurations, and operating systems are covered. For example: (M-Files Corporation, 2018e)

- Web access: user story tests using different browsers
- User story tests done partly in on-premise and partly in a cloud environment
- Desktop: user story testing in different supported Windows OS versions
- Mobile: user story testing of Android story using two different Android devices with different OS versions.

In addition to user stories, there are also other testing tasks. Bug fix testing is done to ensure that fixes work as expected, and no new regression issues found. It is done mostly only larger fixes that are not fixed during user story testing. There is also improvement testing that is very similar to bug fix testing but is used for smaller improvements that are not big enough for a user story. (M-Files Corporation, 2018e)

Once the user stories related to a new M-Files feature have been developed and reviewed, the test manager may decide on final feature testing that will cover testing of the whole feature before releasing the feature as a part of an M-Files release. When there are large features developed that contain several user stories, the test manager may also decide on

final feature testing that will cover testing of the whole feature. (M-Files Corporation, 2018e)

2. Test Automation

Automated regression testing of M-Files covers automated API testing and automated functional UI tests for Windows desktop, web, and mobile applications. The test automation consists of three phases of automated tests that are developer, integration, and regression tests. The purpose of these tests is to improve software quality by executing automated test cases as regression tests on a regular basis. (M-Files Corporation, 2018e)

3. Release Acceptance Testing of M-Files Releases

In release testing, M-Files release is tested extensively using the existing test asset for M-Files system. Both the functional and non-functional tests (e.g. performance and security) are included. Testing at this point includes cross-compatibility tests across different versions of M-Files. (M-Files Corporation, 2018e)

4. Periodical Testing Tasks

In addition, there are some tests that are run periodically, for example quarterly. These tests include compatibility tests with M-Files Add-ons. (M-Files Corporation, 2018e)

Existing test automation

Currently, unit testing is conducted with the NUnit unit testing framework (Poole & Prouse, 2017). NUnit is integrated into TeamCity which is a build management and continuous integration server (JetBrains, 2018). Unit testing is conducted for every build. Test cases are using the M-Files server and client API. (M-Files Corporation, 2018e)

User interface test automation is done with Smartbear's TestComplete software which is a functional automated testing platform that gives testers the ability to create automated tests (SmartBear Software, 2018). The target environments of UI test automation are Windows 7, Windows 8.1 and Windows 10. The target for UI test automation is to execute regression test set for one of the M-Files builds every week. (M-Files Corporation, 2018e)

Web UI test automation is performed with Selenium which is a test automation tool for web applications (Selenium, 2018). Selenium helps controlling browsers automatically and to interact with the web application. Web test automation is targeted for Firefox, Chrome and Internet Explorer browsers. The target for Web test automation is also to execute regression test for each M-Files build. (M-Files Corporation, 2018e)

Automation in mobile testing is used to improve testing of M-Files mobile applications. The mobile test automation by Selenium covers a part of mobile release smoke testing.

The general smoke test set covers the basic features of mobile applications. Mobile test automation is targeted for supported iOS and Android OS devices. The test set can be executed on different M-Files server versions, different OS versions, and different devices. It also covers the portrait and landscape use of the device. Mobile testing is executed for every M-Files stable build against released version of the mobile application. (M-Files Corporation, 2018e)

There exists also non-functional test automation. Performance testing is executed for M-Files minor and major release with appropriate performance testing tools. In addition, security testing is executed for M-Files minor and major releases with appropriate security testing tools, e.g.: (M-Files Corporation, 2018e)

- Web vulnerability scanners (Burp and OWASP ZAP).
- Static code analysis (ESLint).

Testing AI now

Currently, testing of AI systems is concentrated only on the user story phase when some new feature has been added. The testing is done as manual explorative functional testing and as manual quality testing by the testing team. In addition, the AI development team performs some level of quality testing with a smaller collection of documents.

Hence, the testing process is not continuous. Release acceptance testing is meant to be kept compact, and the included manual tests are focused mostly on basic functionality because the tests are meant to be done once a month. Hence, there have not been AI specific tests as part of the release acceptance testing. Also, test automation is missing AI specific testing. That is mostly because AI has been included in the product only recently and there are challenges when defining the right answer to problems solved by AI.

3. PROBLEM

The goal of this thesis is to develop a test automation implementation for M-Files Text Analytics that is part of the M-Files Information Extractor vault application. This chapter describes more precisely Text Analytics and the preliminary phases before the actual implementation work. Additionally, the challenges in developing test automation for Text Analytics are introduced.

3.1 Information Extractor vault application

Information Extractor application is an additional functionality that can be installed on M-Files Server. Its task is to generate metadata suggestions based on the document content. It consists of two services: M-Files Text Analytics and M-Files Matcher. (M-Files Corporation, 2018a)

Text Analytics and Matcher can both generate similar metadata suggestions based on document content, but their main extraction principle is different. M-Files Matcher provides metadata suggestions based on information found in the document by matching it to the vault metadata like the name of an existing customer in the vault. M-Files Text Analytics, on the other hand, provides metadata suggestions based on individual metadata suggestion rules and can produce metadata suggestions that are not based on existing metadata such as a new company name that is not yet an existing customer in the vault. (M-Files Corporation, 2018a)

This thesis focuses on the Text Analytics part. M-Files Text Analytics generates metadata suggestions based on the document content analysis. These suggestions may include, among other things, company names, phone numbers, and email addresses. The suggestions are generated based on a few different types of analyses of the document content. The types of analyses are: (M-Files Corporation, 2018a)

- Comparing input text against a predefined list of items such as company names or places.
- Looking for predefined or configured patterns in the input text such as phone numbers or email addresses.
- Looking for predefined or configured patterns in the file path or name.

For many cases, the most useful information extracted by M-Files Text Analytics is going to be the company names or person names found within the document content. Also, suggested document classes based on document file extensions are often useful. Moreover, M-Files Text Analytics offers a rich set of rules for suggesting other potentially important information. This functionality can be enabled and configured on a case-by-case basis. (M-Files Corporation, 2018a)

M-Files Text Analytics is provided as part of the M-Files Information Extractor intelligence service, and as also mentioned earlier, it is installed as a vault application. Installing the service provides a default configuration to start with. The default configuration includes a few specific built-in and custom rules. These default rules include the following entities: bank, company, country, internet address, email address, organization, person name, phone, and class. These rules provide metadata suggestions based on document contents about the corresponding entities as well as document types (classes) based on the file extensions. (M-Files Corporation, 2018a)

The default configuration, however, does not include any mappings between the suggested terms/entities and the vault metadata structure. Such mappings must be added manually after installation. If suggestion mappings are not specified, the information extracted by M-Files Text Analytics is not used for providing metadata suggestions. (M-Files Corporation, 2018a)

For the purposes of the Text Analytics within the M-Files product, the users are less concerned with recognizing each entity. Rather, they are interested in recognizing only the most relevant entities that are ideally ordered by relevance. In other words, high precision is important so that only relevant metadata suggestions are shown to the user. This is done slightly at the expenses of recall, so the main goal is not to suggest all possible values found in the document. Bad quality suggestions get users to question the quality of the product, and since Text Analytics is a visible part of M-Files, it is possible that the users then start to question also the quality of the whole M-Files product. Hence, the focus is on quality and not on the number of suggestions.

3.2 Testing

Testing of the named entity recognition (NER) system is illustrated in Figure 13. First, a manually annotated datasets are needed. Then, the annotations are compared with the NER system output. In the end, evaluation metrics are calculated based on those comparison results.

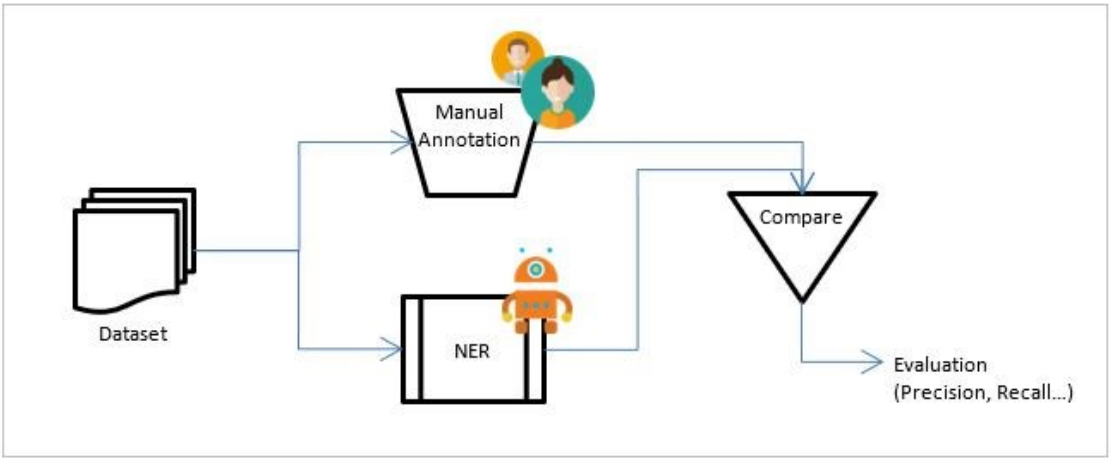

Figure 13. *Testing of the NER system.*

In the context of this thesis, the most central thing is to develop test automation that detects if something has changed between builds. The built-in default rules are used as the basis for testing. Even though it would be good to test as many of those rules as possible, annotation and other test automating activities take time and some level of compromise is needed. Out of the built-in rules, bank and company rules are not encouraged to be used anymore, and they exist mainly to ensure compatibility with older Text Analytics versions. Hence, testing is regularly started with the most used and more complex rules such as organization and person names.

Running time is not critical in test automation. A great number of test files are needed and all of them are gone through for every test case to get good average measures that takes time. Also, formatting the suggested and annotated values to more appropriate forms for comparison takes time because multiple formatting operations are done to all values. The quality of the results is more important than running time in this implementation. Execution time for testing is more important for general NUnit API tests that are run for every build. Shorter execution time allows for faster computation of the results and the subsequent information sharing with the developers and others in case something has broken.

Normally, a maximum number of suggestions per term is set to five in Text Analytics. Suggestions are also in priority order. As previously mentioned, Text Analytics focuses on the quality of suggestions instead of their amount. However, all suggestions that the Text Analytics considers relevant are considered in this test automation implementation. It increases the maximum number of suggestions so that all evaluation metrics can be calculated rationally. This is necessary because normally Text Analytics suggests only the five most relevant terms instead of all the organization or person names found in the document which would distort the evaluation metrics.

3.2.1 Challenges

The first challenge from the testing perspective is to obtain real data. It is understandable that M-Files customers are neither willing to share their partly sensitive real data for testing purposes nor eager to allow developers even to take look at it. Hence, the nature of the documents needs to be guessed. Additionally, M-Files is used by different type of companies for several different purposes, so testing cannot concentrate only on specific types of documents.

In principle, the testing dataset should be independent of training and validation datasets and unknown to the developer. It is usually hard to get good quality real business documents as input to training and testing. In addition, the annotation of many documents requires a lot of time. Due to these reasons, the same dataset is used both for developing and testing. Furthermore, the developer gets to know the testing dataset as well when the tester shares the test results with the developer. This is made possible by the fact that data is not used for training in Text Analytics. The possible shortcomings caused by the compromise on the dataset independency principle must be regarded when evaluating the effectiveness and accuracy of test automation in this case.

Another thing to consider is language. Text Analytics should work with multiple languages like M-Files in general, so it should also be tested with those languages. Problem with this is getting and annotating test files for all the languages. An annotator should be familiar with the language that she/he is annotating. Since a great number of files are required for each language, multiple annotators are also required. English was chosen as the language for this implementation work due to the language skills of the annotators.

An additional challenge is Optical Character Recognition (OCR). OCR is a technology that enables converting different types of documents, such as scanned paper documents, PDF files, or images captured by a digital camera into machine-readable i.e. editable and searchable text data. Text Analytics uses OCR to get the text from some of the files. If OCR does not work correctly, the entity extraction of Text Analytics becomes more complicated. Although OCR is not directly part of this test automation implementation, it should be taken into consideration because occasionally it can be the reason for the bad results of Text Analytics. If the text is not recognized correctly, named entity extraction will not be able to work properly.

The greatest number of challenges is caused by comparing metadata suggestions and annotation results. The reason is that the same organization or person name can be written in many ways, which means that the annotation file can include the same name several times in different formats. It is enough that only one of the options is included in the metadata suggestions. Even though other options are not suggested, it should not have a negative effect on the results. All the formats of the same name are separated in the annotation file, and the challenge is to determine which ones refer to the same thing.

Organization names can, for example, be written as Auto By Cat Corporation, Auto By Cat Corp., Auto By Cat, or ABC. In addition, there can be multiple names. One problematic abbreviation is dba which means “doing business as”. It can be included in the name as for example in Cyrus Networks, LLC dba CyrusOne.

Person names can also be in different formats. For example, Maurice R. "Hank" Greenberg, Maurice R. Greenberg, Maurice Greenberg, M. Greenberg or Dr. Greenberg. It is expected that the metadata suggestions are unique enough so that different names can be clearly distinguished. Since separate first and last names are not considered unique enough, they are neither annotated nor expected to appear in the metadata suggestions separately. In addition, the suggestions are expected to be in a basic format, and hence, courtesy titles and designations, e.g. Dr., Mr., Esq., and Ph.D. are excluded from the annotations.

Comparing the results, it needs to be defined when metadata suggestion is considered as being correct. In addition to different formats in the names, a challenge is caused by partly correct answers. For example, the annotated correct suggestion is “Tampere University of Technology”, but the resulting suggestion turns out to be “Tampere University”. The question is whether the suggestion is then incorrect, correct, or partially correct. In this implementation, partially correct answers are not accepted and are considered as being incorrect.

The annotation file can be in different formats depending on the annotation tool. The format can influence the ways of parsing the annotations i.e. the correct answers. For this thesis, Brat annotation tool is used. It has its own unstandardized file format. Brat has its shortcomings which is why in the future another tool with a different format might be used. Hence, the implemented test automation solution should take into consideration that there might be a change in the annotation file format. The easiest approach is to implement parsing separately from the other implementation which will make it easier to modify without changing the solution code.

3.2.2 Annotation

Annotations are needed to represent the desired, correct output. They are used as the ground truth to which the results produced by Text Analytics are compared.

In the first phase, only person names and organization names are annotated. Then possibly at a later phase, also other entities are annotated. Those are the ones that M-Files' Text Analytics recognizes: country, internet address, email address, and phone number. Subsequently, the annotated organizations are divided into subcategories such as org-Government, org-Commercial, org-Educational, org-Entertainment, org-Non-Governmental, org-Media, org-Religious, org-Medical/Science, and org-Sports to describe the nature of the organization. For the moment, all organizations are handled as one because there is

only one built-in rule for organizations. Subcategories are more for future development. There is also a tag "Ambiguous" that can be used to tag something that looks like it might be an entity but there is not enough context to determine exactly what kind of entity it is. Generally, it is better to tag something as ambiguous rather than not to tag it at all in case of uncertainty. In all, these annotations are meant to validate Text Analytics i.e. metadata. So the most important questions to consider when deciding whether to include a term in metadata or not:

- "Is this something a user might expect as metadata?"
- "Is this something that could be matched with (or added to) an object in the vault?

For this case, annotations are started with 150 English business documents that are obtained from public sources, and as soon as possible, the document collection is expanded with other 150 English public source business documents. There are three annotators for each document, and after the individual annotations are done, the resulting annotations are combined as one.

Annotations are made with the open source tool Brat. Every annotator has a folder for the documents that he/she is annotating. Unfortunately, Brat does not support combining the annotations of different annotators, so it has to be done manually.

Nancy Chincor's "Named Entity Task Definition" is well-suited to serve as an annotation guide, and it is used as one in this thesis (Chinchor, 1997). Since that only has organization as one category, a separate guideline is needed for organization subcategories. The document "ACE (Automatic Content Extraction) English Annotation Guidelines for Entities" by Linguistic Data Consortium at the University of Pennsylvania from the year 2008 is used for subcategory annotation (Linguistic Data Consortium, 2008). ACE is a comprehensive annotation standard that aims to consistently annotate entities, events, and relations within a variety of documents. The ACE standard was developed by the National Institute of Standards and Technology in 1999 and has evolved over time to support different evaluation cycles, the last evaluation having occurred in 2008.

3.2.3 Used evaluation metrics

The chosen evaluation metrics may have a significant effect on the result. They need to be suitable for evaluating the function of the named entity recognition.

Given an input text or a collection of texts, the expected output of an information extraction system can be defined very precisely. Therefore, precision and recall metrics can be adopted from the information retrieval research community for that purpose. They are used to measure the effectiveness of the system from the user's perspective. Recall evaluates to which extent the system produces the correct output whereas precision evaluates the extent of producing only the correct output. Thus, recall and precision can be seen as

measures of completeness and correctness respectively. (Poibeau, Saggion, Piskorski, Yangarber, & (eds), 2013, p. 28)

The metrics of recall, precision, and F1 Score can also be used to evaluate NER systems. For named entities, the entity itself rather than the word describing is the unit of response. It should be noted that named entity tagging has a segmentation component which is not present in tasks like text categorization or part-of-speech tagging, and it causes some problems with the evaluation. For example, a system that labeled American as an organization and American Airlines not as one would cause two errors, a false positive for not-entities and a false negative for organizations. In addition, using entities as the unit of response but words as the unit of training means that there is a mismatch between the training and test conditions. This mismatch needs to be fixed for evaluation. (Jurafsky & Martin, 2017, pp. 353-354)

For this thesis, precision, recall, and the F1 Score that is computed from them are considered useful. To be able to compute precision and recall, the values for true positives (TP), false positives (FP), and false negatives (FN) need to be computed.

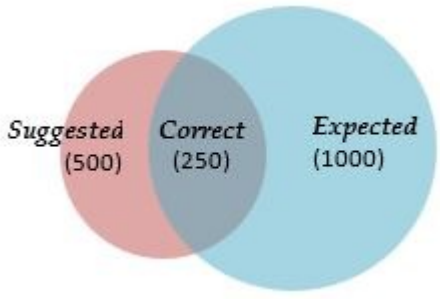

Figure 14. *Venn diagram of suggestions and expected values.*

Another way to illustrate true positive, false positive and false negative in this context is shown in Figure 14. Suggested values illustrate the given metadata suggestions and expected values illustrate the expected metadata suggestions or annotated values of the tested term/entity. The correct values express true positives or correctly suggested values. Suggestions that were not correct are false positives and expectations that were not suggested are false negatives.

4. SOLUTION

This chapter describes the implementation of test automation. First, the solution environment and test data management are explained. Then, the used test cases are presented. Finally, the actual implementation is described. The implementation includes the NUnit testing framework, the process of getting suggestions and annotations as well as their comparison process, and the process of computing the evaluation metrics.

4.1 Solution environment

Visual Studio integrated development environment and the NUnit 3 Test Adapter that allows you to run NUnit 3 tests inside Visual Studio are used for developing the test automation. C# that is suitable with Visual Studio and NUnit is used as the programming language. All tools used in this thesis are commonly used in M-Files and are also used in the existing test automation. Hence, using those tools is simple because employees are already familiar with them and the new test automation implementation would then suit well as part of the other M-Files related implementations.

M-Files already has an existing testing environment that has been built on top of NUnit for running API integration tests. Testing is done through the API to test the entire M-Files, which means that a new vault including users and preferred settings needs to be created for the test.

The existing code makes the implementation work for this thesis slightly easier since everything does not have to be built from the beginning. The new test automation will utilize the existing code, but it will be implemented as an independent Visual Studio project. Necessary dependencies to the other M-Files code are added to the new project. These dependencies are M-Files System and M-Files API NUnit Framework projects.

The previous NUnit implementation, M-Files API NUnit Framework project, supports writing clean and proper test cases. The previously created code is mostly used for building and cleaning the test environment as well as creating and removing M-Files object used in the tests. Because of the reuse of the code, it is not necessary to concentrate on other M-Files specific API usage in this thesis implementation. Instead, the implementation work can concentrate mostly on Text Analytics specific things.

4.2 Test data management

Many external files are used in the implementation. This is because multiple test files are needed to get statistically meaningful average results. Each test file needs to have a corresponding annotation file that serves as the ground truth for the metadata suggestions

given to the test file. Test and annotation files need to stay the same all the time so that old results are comparable to new ones. In addition to files used in testing, the result files are needed to possibly inspect more closely what data is behind the results so that possible issues are easier to track.

There are separate folders under the visual studio project for the test files and result files. The test files folder includes all 150 business files that are used in the tests. Inside the test file folder, there is a folder for the annotation files. All files are part of a project to make them easily available when running the tests. When the tests are integrated into TeamCity, they are run in different environments, like different cloud instances. Hence, test files need to be available in every environment.

There are two options for saving the test files. They can be either in their original format (such as pdf, doc or docx) or in the text format. If they are in the original format, the space needed is 61.3 MB, and if they are in the text format, the space needed is only 7.33 MB. It means that the original format requires about twenty times more space than the text format. It can be argued whether the test files should be in their original format or in the text format. The original format considers things as a whole, and if, for example, OCR changes and affects results, it will be noticed. On the other hand, the text files do not require so much space and changes in things that are not directly related to Text Analytics do not affect so much. In this case, the test files are saved in the text format because the original format requires much space. Size matters because the files are part of M-files code base that already includes the whole M-files code which is quite huge by itself.

There are also different options for saving the annotation files. If the annotation files are in Brat annotation tool format which is a sort of a text file, their size is 191 KB. If the annotation files are converted to more easily handled JSON format, their size is 49.4 KB. The annotation files in JSON format are almost four times smaller than in Brat format. The size difference is not significant because both are quite small. The more important thing is that annotations are easier to get from the JSON format and parsing needs to be done only once when converting and not every time when running the test automation which saves time. Test automation should be as fast as possible so that the results are obtained fast, and if there were problems, they could be solved quickly.

```
{
  "File": "06-26-17_Special_Meeting_Minutes.pdf",
  "Entities": {
    "Person": ["Tim Kauppi", "Mary Hess"],
    "Organization": [ "City of Hoyt Lakes", "League of Minnesota Cities", "Mesabi East School District",
                      "Gornick's Nuisance Wildlife", "CITY OF AURORA COUNCIL" ],
    "Ambiguous": ["Town of White"]
  }
}
```

Figure 15. *A JSON formatted annotation file.*

An additional implementation is created to transform the Brat annotation tool generated annotation files into JSON format. Brat's annotation files do not use any specific format

like JSON or XML. To get those annotation files to more easily handled format, they are converted to JSON. An example JSON file can be seen in Figure 15. In the JSON format, all annotated values are already in a list according to the entities, so annotations are easy to get for a specific entity. It saves time in the test execution when annotations do not need to be parsed from Brat's file format. Even though organizations were divided into subcategories in the annotation phase, all of them are now combined as one organization term because there are no subcategories in Text Analytics. The test automation implementation uses directly annotation files in the JSON format. JSON conversion needs to be done only once, and after that JSON format annotation files are saved as part of the test automation implementation.

Inside the results folder, there will be a separate CSV result file for every test case after executing the test. The result file system can be changed when the test runs are integrated into TeamCity. The target is to get the result files included in the TeamCity results so that the old results file could be monitored for every test run via TeamCity.

4.3 Test cases

There is one test case for every Text Analytics term that is to be tested. When only one term is configured at the time it is easier to interpret the results and easier to see what the possible issue is. There are several terms that can be used to give suggestions in Text Analytics: bank, company, country, internet address, email address, organization, person name, phone number, and class. In this case, only the organization and person name terms are tested.

Each test case can be run separately, or all test cases can be run together sequentially in one test run. When multiple test cases from the same class are run, they use the same test environment which means that vault creation and other setup procedures need to be done only once. The test environment creation and destruction take quite a lot of time and, hence, requires a lot of extra time when carried out multiple times. A common test environment means that the test cases can affect each other. Therefore, it is important that the test cases are independent and leave the test environment in the same state that it was at the beginning which is what test cases should generally always do. In this implementation, there is not such a big concern because every test case configures Text Analytics at the beginning of the test to be suitable for its purposes and uses only M-Files objects that it has created itself. In addition, even though every test case uses the same test files, those files are only used to get metadata suggestions and, therefore, are not modified.

In a single test case, the comparison of metadata suggestions and annotations is done for 150 documents to prevent the results of a single document of having too much influence. It is not reasonable to base the results only on a single document because different documents could easily produce varying results.

4.4 Implementation

The implementation starts by the initialization of the test environment and configuration of Text Analytics to give metadata suggestions for a specific term. Next, an M-Files object is created and a test file in turn is added to it. Then, the metadata suggestions are retrieved for it, and the received metadata suggestions are compared with annotations or the right answers. The results are written into a result file. When proceeding to the next test file, the previous M-Files object is removed. This process is repeated for all test files. In the end, results from the separate files are combined, an average of the results is calculated, and the testing environment is removed.

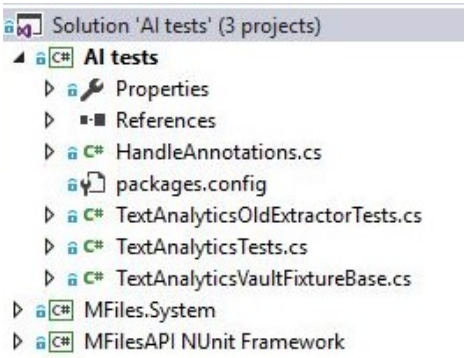

Figure 16. *Solution structure tree.*

Structure of the solution can be seen in Figure 16. TextAnalyticsVaultFixtureBase takes care of creating and destroying the test environment. TextAnalyticsTests includes all test cases. TextAnalyticsOldExtractorTests can be used to run the same test cases but using an older version of Text Analytics. This is mostly for curiosity purposes to see how Text Analytics' results have changed from the first version. HandleAnnotations includes help functions to get annotations for the test file in turn, comparing suggestions with annotations and calculating evaluation metrics. In addition to those implementation files, the solution includes folders for the test, annotation, and result files that are not visible in the figure.

4.4.1 NUnit

Code implementation is done on top of NUnit that is a unit testing framework for all .Net languages. NUnit is written entirely in C# and has been designed to take advantage of many .NET language features like custom attributes and other reflection related capabilities. NUnit tests can be written using any .NET language that supports attributes. Attributes are used to indicate test classes and methods and to modify their behavior in various ways. For example, TestFixture attribute marks a class that contains tests, and Tests attribute is used to mark a method inside a TestFixture class as a test. NUnit uses assertions to test an actual value against one or more constraints. The assertions produce either a

successful test or a failure. The test context is used in NUnit by the user code to acquire information about the test and its environment. (Poole & Prouse, 2017)

To avoid creating everything from scratch, the previous NUnit implementations in "M-Files API NUnit Framework" project are used. Those include ready FixtureBases, test vaults, helper functions, etc. that can be used when creating the test environment. FixtureBase includes the base classes that provide the test users and the access to vault for the actual test fixture. A fresh vault and users are automatically created for the test fixture. All new test fixtures should inherit from one of the provided fixture bases.

New FixtureBase called TextAnalyticsVaultFixtureBase is created when inheriting basic FixtureBase from the M-Files API NUnit Framework project. It includes functions to set up and clean test environment. Setting up includes creating M-Files vault with users and installing the information extractor vault application and cleaning that includes, for example, destroying the used vault. Created FixtureBase also includes a function to configure Text Analytics with the wanted term. When configuring Text Analytics, it needs to be considered that configuration includes a maxNumberOfSuggestionsPerTerm item, and it is set to five by default. If that item is not changed, it drastically affects the results because all suggestions are not considered. Therefore, the maximum number of suggestions is increased to 50 which is thought to be an appropriate upper limit for the number of suggestions per file.

There is a separate NUnit test case for every tested, marked entity. When running a test case, Text Analytics is first configured to give suggestions using the tested term, either the organization or the person name. Next, all 150 test files are gone through. For each file, an M-Files document is created, and metadata suggestions are requested for it. Then, the given metadata suggestions are compared with annotations, and the evaluation metrics are calculated. The results are written to a CSV file, and the created M-Files object is removed before creating a new one for the next test file. After all test files have been evaluated, the average of the evaluation metrics is calculated. Those metrics are then used in assertions to determine if the test case passes or fails.

4.4.2 Comparison

The correctness of metadata suggestions is figured out by comparing given metadata suggestions with the annotated named entities. When doing the comparison, things like different formats of names as described previously in this section need to be considered.

Before conducting the actual comparison, the suggested and annotated values are normalized and formatted to make it easier to compare them with each other. At first, all values are changed to lowercase so that the comparison is case insensitive and extra words like “a”, “an” and “the” are removed.

In addition, special formatting is done to values based on the entity they belong to. If the value's entity is a person name, initials and nicknames in quotes between the first and the last name are removed. For example, Maurice R. Greenberg or Maurice "Hank" Greenberg is formatted to the form Maurice Greenberg. If the value's entity is an organization, more formatting needs to be done. Most commonly used abbreviations in organization's names are replaced with the actual words according to a dictionary that is shown in Figure 17. The endings ("oy", "inc", "ltd", "llp", "llc", "lc", "co", "corp", "incorporated", "limited") that are usually used in organization abbreviations are removed. Also, commas and dots are removed. As the final phase in formatting, all extra spaces and doubles are removed.

```
// Dictionary of organization abbreviations.
private static Dictionary<string, string> org_abbreviations = new Dictionary<string, string>()
{
    {"amer.", "america" },
    {"ass'n", "association" },
    {"bd.", "board" },
    {"comm'n", "commissiont" },
    {"comm.", "committee" },
    {"cmty.", "community" },
    {"co.", "company" },
    {"cong.", "congress" },
    {"corp.", "corporation" },
    {"cnty.", "county" },
    {"dep't", "department" },
    {"dist.", "district" },
    {"grp.", "group" },
    {"inc.", "incorporated" },
    {"inst.", "institution" },
    {"ins.", "insurance" },
    {"int'l", "international" },
    {"ltd.", "limited" },
    {"serv.", "service" },
    {"svc.", "services" },
    {"univ.", "university" }
};
```

Figure 17. *A dictionary that includes abbreviations commonly used in organization names and the corresponding whole word.*

With persons' names, entity comparison is more straightforward. A suggestion is thought to be correct if the same value is found in the annotation file either as a specific entity or an ambiguous entity. If the suggestion is not found in the annotations, it is thought to be wrong.

With organizations, entity comparison becomes more complicated. In the beginning, a list including abbreviations is created from the annotated organization names. When creating abbreviations, organization names with short words like "of", "and" or "for" which are not usually used in abbreviations are abbreviated both with and without those words. The same is done for words like "international", "company" and "corporation" that may or may not be part of an abbreviation. Then, the first character of every word in the organization name is selected and they are combined to form the abbreviation. Abbreviations with only one character are excluded. Abbreviations that are formed in a different manner than using only the first characters are not included because that would be too challenging. For example, Siemens Corporation is abbreviated as SC and not as SiCo as

it officially should be. Next, the additional words like "international", "company" and "corporation" are removed from all suggested and annotated values because an organization name can sometimes show up with or without those words. This will help reducing false negatives for cases when an organization name is written without those additional words because also those will be now covered.

A suggestion of an organization name is correct if the name is found either from the list of specific entities or ambiguous entities in the annotation file or from a derived list of abbreviations. Otherwise, it is incorrect. A missing suggestion is a specific annotation that was not suggested.

Suggested	Annotated	Ambiguous
EC	EC	
EC	Example Car	
EC	Example Car Company	
EC		EC
EC		Example Car
EC		Example Car Company
Example Car	EC	
Example Car	Example Car	
Example Car	Example Car Company	
Example Car		EC
Example Car		Example Car
Example Car		Example Car Company
Example Car Company	EC	
Example Car Company	Example Car	
Example Car Company	Example Car Company	
Example Car Company		EC
Example Car Company		Example Car
Example Car Company		Example Car Company

Figure 18. *Different organization name combination scenarios.*

Figure 18 illustrates different organization name scenarios. The suggested column shows the organization names suggested by Text Analytics. The annotation column represents the organization names annotated as specific entities and the ambiguous column as ambiguous entities. Each row in the figure represents one possible combination of these values. The colors show which combinations will result in the suggestion being evaluated as correct (green) and which not (red).

There are a few of combinations that result in the suggestion evaluated as incorrect. First, it is quite unrealistic for a suggestion of the whole name of an organization appear if only abbreviations of it exist in the text. Hence, the cases where the suggestion is the whole name, and the annotation is the abbreviation are not considered. Second, if an abbreviation of a value marked as ambiguous is suggested, the suggestion is evaluated also as incorrect. That is because ambiguous entity classification is not meant to be used a lot. It is only used for those named entities for which the annotator is not sure whether they should be suggested or not.

4.4.3 Evaluation metrics and result file

After the comparison is done, its results are used to calculate the evaluation metrics. Evaluation metrics help in noticing if the functionality of a tested software has been significantly changed. It is important to note that when counting the different metrics, multiple forms of organization names are combined to one so that different forms of the same name would not cause distortion to the results. For example, abbreviations are removed for all those organization names whose whole name is also annotated.

The count of correctly made suggestions is true positive (TP) and the count of suggestions that are found in annotations neither as a specific nor as an ambiguous entity is false positive (FP). The count of annotated values for a specific entity that is not suggested is false negative (FN). Based on TP, FP and FN, precision, recall and F1 Score are calculated to the single test file.

When calculating precision, recall, and F1 Score certain special cases need to be considered. If the TP, FP, and FN are all 0, the precision, recall, and F1 Score are 1. If TP is 0, and one of the two other counters are larger than 0, the precision, recall, and F1 Score are 0. (The Research Group Agile Knowledge Engineering and Semantic Web (AKSW), 2017)

The result file is a CSV file that is meant to help the manual inspection of the results. There is a separate result file for every test case. The result files help discover reasons why evaluation metric values are what they are and whether the reason is in the test automation implementation or in the Text Analytics implementation. The result files can be used to discover the patterns of situations where Text Analytics does not perform that well. This can then help improve the Text Analytics implementation. An example of a result file is shown in Figure 19.

File	Suggestions	Annotations	Ambiguous	True Positives	False Positives	Flase Negatives	TP	FP	FN	Precision	Recall	F1 Score
06-26-17_	Hoyt Lakes,Ma	Mary Hess,Tir	Town of Wh	mary hess,tim h	hoyt lakes		2	1	0	0,666667	1	0,8
1214_-_be	Ignacio Bernal	Ignacio Berna		ignacio bernal			1	0	0	1	1	1
2009_012							0	0	0	1	1	1
427_Expar	Bert Clark,Bob	Bert Clark,Bol		bert clark,bob r			5	0	0	1	1	1
702-letter-	John Conyers	Bob Goodlatt		john conyers		bob goodlatte	1	0	1	1	0,5	0,666667
BBAM201	Charles Ramir	Ben Blamire,C		charles ramirez		ben blamire,dee	13	0	8	1	0,619048	0,764706
Billing.pdf	Bill Search,Edv	Katherine Wo	Accounts Re		bill search,edw	katherine wojto	0	2	2	0	0	0
Business-P							0	0	0	1	1	1
filedownlo	Amy Crites,An	Alena Zachary	IISD,IISD Boa	amy crites,andı	bennett woods	alena zachary-rc	21	1	7	0,954545	0,75	0,84
MOUAmrc	Cornelius Van	Bob Benmosc	AIG Board	cornelius vande		bob benmosche	4	0	6	1	0,4	0,571429
ngolist.pdf							0	0	0	1	1	1
TCC_2017	Andrew Grant	Andrew Grant	Bicycle and F	andrew grant,a	beulah church,	andy grzymski,ar	39	6	9	0,866667	0,8125	0,83871
UCM2296	Adam Clark,Ar	Adam Clark,Ad	AC,Advisory	adam clark,and	ann meeker-o'	amy allina,bill va	41	9	21	0,82	0,66129	0,732143
SUM							127	19	54	0,869863	0,701658	0,776758
AVG							9,769231	1,461538	4,153846	0,869837	0,749449	0,785666

Figure 19. *Person name result file example.*

There is a separate line for every test file in the result file. That line includes the name of the test file, suggested metadata suggestions in original format, annotated values of a specific entity in the original format, annotated values marked as ambiguous in the original format, true positive values after formatting, false positive values after formatting, false

negative values after formatting, and calculated evaluation metrics values for TP, FP and FN, precision, recall and F1 Score. All values that are used in the comparison are in sorted order in the result file. It is then easier to see the differences manually when the values are in a list format.

The result file also includes a summary and average lines. The summary line includes the total number of TP, FP, and FN from all test files and precision, recall and F1 Score calculated from those counts. The average line includes the average of TP, FP, FN, precision, recall and F1 Score across all test files.

4.4.4 Pass or fail decision

Verification is the most important step in writing a good test case. The test automation can only look for issues it has been instructed to look for. All other issues will be missed unless they cause a catastrophic failure.

In NUnit, verification is commonly done with the help of an Assert class. It has static methods that can be used to make generic assertions such as equality comparisons on the state of the environment. If the specified assertion does not hold, an exception with a descriptive error message is thrown by the NUnit engine.

There are specific threshold values for precision, recall, and F1 Score for every test case for passing. Before the comparison of the evaluation metrics, it is ensured that Text Analytics has given at least some metadata suggestions by ensuring that total number of given suggestions (total number of TP + total number of FP) for all test files is more than zero. After ensuring that, the averages of the metrics are compared with the thresholds of the test case. If the average precision, recall or F1 Score is lower than the threshold, the test fails. Ensuring that Text Analytics gives suggestions is essential because if suggestions are not given at all, precision, recall and F1 Score will all be equal to one. Thresholds are there to alert if something has changed. If the accuracy of Text Analytics has clearly decreased, the problem needs to be solved. The threshold should be defined separately for every test case because there can be significant differences in used metrics between them. For example, the precision of a person name is over 0.85 and the precision of an organization name is over 0.6.

There are different options that can be used to define the threshold values. One option is to set the thresholds below the current evaluation metric values. Then it is easily observed if the quality of Text Analytics goes worse because the decrease will cause the test to fail. Another option is that the threshold values set a standard to be achieved. In that case, Text Analytics could be thought of as a shippable product to customers and the standardized threshold values would represent the level of the aimed quality. The problem with this option is that the current Text Analytics might not achieve those thresholds and the tests would fail until some improvements were done. If the tests fail until improvements are

done, it is not easily observable whether the modifications made the current solution even worse. Therefore, threshold values are set to be just under the current performance level are used in this implementation so that a test case would fail only if the implemented changes lowered the quality of the solution.

4.5 Integrating test runs to TeamCity

For the new test automation to be most useful, it needs to be a part of the continuous integration pipeline so that the tests can be easily run and the results can be easily seen. A tool used to build management and continuous integration in M-Files is TeamCity. It can be used to set the tests to run automatically, for example, for every stable build or release build, and the test results would be easily visible.

TeamCity from JetBrains is a continuous integration server that supports a variety of different version control systems and build runners. It can automate code analysis, compiling and testing the process while providing instant feedback on build progress, problems, and test failures through a web interface. In addition, it enables running multiple builds and tests under different configurations and platforms simultaneously. It also has a build history functionality with customizable statistics on build duration, success rate, code quality, and custom metrics. (JetBrains, 2018)

It is possible that the test runs will not be integrated into TeamCity within the scope of this thesis. Regardless, getting the test runs to TeamCity should be quite easy because the tests are implemented using the NUnit framework. In addition, the previous NUnit test cases are already running in TeamCity. TeamCity saves the test run history, and it should be also possible to save the result files as part of the test run results in TeamCity which means that result files of previously ran tests can be viewed afterward.

5. FUTURE DEVELOPMENT

There are multiple ways to develop the implemented solution forwards. The main ways are either expanding the testing of Text Analytics or expanding the solution to test other AI-based solutions.

One way to expand the testing of Text Analytics is adding more test files. This would make the results are statistically more reliable. However, that is not probably the best approach to take because it requires more annotation work and does not offer significant additional benefit. This approach would also create another problem. Addition of test files makes it impossible to directly compare the old results with the new results even though results should be co-directional.

Another way to expand the testing of Text Analytics is to add additional Text Analytics' terms/entities. For example, new test cases for email and phone number terms could be created. These terms could also turn out to be more common in customer data. The results of these terms' metadata suggestions should be better because email addresses and phone numbers are usually expressed in a clearly defined format. Thus, it is easier to create a regular expression that finds those, and the probability of false positives would be lower. This way would, however, require more manual annotation work. Regardless, the same test files could be used by adding the new entities and annotating those. This would require only very minor code changes.

Yet another way to expand the testing of Text Analytics is adding tests for different languages to test that Text Analytics can find organizations and person names also in other language documents than English. That would require gathering new test files that need to be manually annotated using the same entities as in this implementation. Probably the next language would be Finnish because that is the most common language of the annotators and the customers using M-Files. The Finnish language brings new challenges to the annotation. In Finnish, not all words are in the basic format. Instead, they can be conjugated in multiple ways. For example, the basic form of the word "yritys" can be conjugated as "yrityksellä" or "yrityksen". If the word cannot be annotated in its basic form, it creates many issues in the comparison phase. This approach would, however, require very minor code changes. Only a new test case that uses a different folder as the test file source should be added.

In general, information extraction from English texts appears to be somewhat easier than from other languages that exhibit more complex linguistic phenomena, e.g., richer morphology, and freer word order. Similarly, information extraction is easier from grammatically correct texts, such as online news, than ungrammatical short messages, such as tweets and blog posts. (Poibeau, Saggion, Piskorski, Yangarber, & (eds), 2013, p. 31)

In addition to expanding the testing of Text Analytics, the solution can be expanded to test other AI-based solutions. One option would be, for example, M-Files Repository Sensor vault application. M-Files Repository Sensor searches for Finnish personal identity numbers from documents and if it finds those the document is marked as sensitive. The testing of the Repository Sensor would again require some new test files that would include personal identity numbers. The objective would be to mark files that included personal identity numbers by, for example, annotating those numbers from files. Then it would be known if a file included a personal identity number. General Data Protection Regulation (GDPR) of European Union restricts usage of real personal identity numbers. One way to solve this problem is to use temporary personal identity numbers that are in the same format as real personal identity numbers. This kind of expansion would require somewhat more code changes. For instance, a New FixtureBase that installs M-Files Repository Sensor vault application and configures it needs to be created. The file format of the results should be changed, and the comparison algorithm needs to be slightly modified.

6. CONCLUSIONS

The target of this thesis has been achieved. Test automation to ensure the working and to measure the quality of Text Analytics has been implemented. For now, it can be run manually, and in future, it will be added to be part of the continuous integration pipeline so that it is automatically run for the wanted versions of M-Files.

Running time of the whole test automation is around 5 minutes and 20 seconds. That can vary slightly depending on the environment. Running an individual test case takes around two minutes. Running a test case of the person names takes around 105 seconds and a test case of organization names takes around 130 seconds. It is understandable that running an organization name test case takes longer because they include more formatting and abbreviations are considered in those tests. Building and removing a test environment takes around 70 seconds. The overall running time is quite sensible since the creation and removal of the test environment takes its time and an individual test case needs to go through 150 test files.

Comparison of suggestions and annotations were the hardest part of the implementation. All different scenarios cannot be considered. Some scenarios are very hard or even impossible to implement and some are thought to be very rare and hence, not worthwhile. For example, scenarios not included in this work were those where:

- The whole name of an organization is suggested, but only the abbreviation is annotated.
- An abbreviation of an organization is different than the first character from every word in the organization's name.
- dba (doing business as) is used that means that multiple names refer to the same organization.
- The first or the last name is substituted with an initial.
- The suggestion is only partly correct, and it is thought to be wrong.
- Specialties or organization name endings are in other languages than English.

Substituting the first or the last name with an initial means, for example, writing John Smith as J. Smith. In this implementation, these two formats cannot be stated to mean the same thing. In the case of partly correct suggestions where the correct organization was Tampere University of Technology, and Tampere University was suggested, the suggestion is considered as FP and the annotated organization is calculated as FN.

When exploring the result files, specific problematic points where Text Analytics does not perform so well can be found. For example, if multiple names like two organizations or person name and organization name are directly one after the other, Text Analytics often thinks that those belong together and proposes the combination as an organization

name or splits them incorrectly. Furthermore, Text Analytics analyzes only plain text and during this text extraction step which means that most, if not all, structure (tables, styles, etc.) is lost. An example where this happened was a document that included a table of company names. When the document was converted to plain text, there were several company names one after the other, which caused problems as mentioned above.

With person names, Text Analytics has also clearly problematic points. Two-part location and city names cause false positives so that those are suggested as person names even though they are not. False positives were, for example, Hong Kong, Houston Texas, Alberta College, Elliot Lake, Forest Hills, Beulah church, John Street, and Wesley Chapel. Last names starting with Mc are not suggested most of the times causing false negatives. Text Analytics uses default configuration in the test automation, so it is optimized to find English names and, therefore, names from different cultures are not recognized even though those are in English documents. For example, Asian and Middle-Eastern names, such as Chen Yu Hua, Shoon Hau Tsin, Nasim Abdullah, Ekram Hossain, and Haider Al-Saidi, caused false negatives.

REFERENCES

Brat. (2018). *mini-introduction to brat*. Retrieved June 14, 2018, from http://brat.nlplab.org/introduction.html

Cheatham, T. J., Yoo, J. P., & Wahl, N. J. (1995). Software Testing: A Machine Learning Experiment. *ACM Conference on Computer Science 1995* (pp. 135-141). Nashville: ACM. doi: 10.1145/259526.259548

Chinchor, N. (1997, September 17). *MUC-7 Named Entity Task Definition.* Retrieved July 5, 2018, from https://www-nlpir.nist.gov/related_projects/muc/proceedings/ne_task.html

Crispin, L., & Gregory, J. (2009). *Agile testing: a practical guide for testers and agile teams* (1st ed.). Pearson Education, Inc.

Haikala, I., & Mikkonen, T. (2011). *Ohjelmistotuotannon käytännöt* (12 ed.). Talentum Media Oy.

Hurwitz, J., & Kirsch, D. (2018). *Machine Learning For Dummies* (IBM Limited Edition ed.). John Wiley & Sons, Inc.

Indurkhya, N., & Damerau, F. J. (2010). *Handbook of natural language processing* (Second edition ed.). Chapman & Hall/CRC.

ISO/IEEE/IEC. (2013). *ISO/IEC/IEEE 29119-1: Software systems engineering - software testing - part 1.*

Japkowicz, N., & Shah, M. (2011). *Evaluating Learning Algorithms: A Classification Perspective.* New York, USA: Cambridge University Press.

JetBrains. (2018). *TeamCity*. Retrieved October 8, 2018, from https://www.jetbrains.com/teamcity/

Jurafsky, D., & Martin, J. H. (2017). *Speech and Language Processing - An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition* (Third Edition draft of August 28, 2017 ed.). Retrieved July 22, 2018, from https://web.stanford.edu/~jurafsky/slp3/ed3book.pdf

Kapetanios, E., Tatar, D., & Sacarea, C. (2014). *Natural Language Processing: Semantic Aspects.* CRC Press, Taylor & Francis.

Kasurinen, J. P. (2013). *Ohjelmistotestauksen käsikirja* (1 ed.). Saarijärven Offset Oy.

Linguistic Data Consortium. (2008, June 13). *ACE (Automatic Content Extraction) English Annotation Guidelines for Entities.* Retrieved July 5, 2018, from https://www.ldc.upenn.edu/sites/www.ldc.upenn.edu/files/english-entities-guidelines-v6.6.pdf

M-Files Corporation. (2017, August 22). *M-Files Acquires Apprento to Bolster Artificial Intelligence Capabilities in Next-Generation Intelligent Information Management Platform*. Retrieved August 10, 2018, from https://www.m-files.com/en/press-release-m-files-acquires-apprento

M-Files Corporation. (2018a). *Configuring the M-Files Information Extractor Intelligence Services.* Retrieved July 5, 2018, from https://kb.cloudvault.m-files.com/Default.aspx?#3ECA226F-7B54-428B-B539-DE443E6134EC/object/132ECE4A-6128-4828-9938-77626601EE2C/latest

M-Files Corporation. (2018b). *Contact us*. Retrieved July 13, 2018, from https://www.m-files.com/en/contact-us

M-Files Corporation. (2018c, February 14). *Lehdistötiedote 14.2.2018*. Retrieved July 13, 2018, from https://www.m-files.com/fi/Press-Release-M-Files-2017-Momentum

M-Files Corporation. (2018d, April). *M-Files 2018 User Guide*. Retrieved July 5, 2018, from https://www.m-files.com/user-guide/latest/eng/

M-Files Corporation. (2018e). Test Strategy of M-Files v2.3.

Mills, M. (2016, February 23). *Artificial Intelligence in Law: The State of Play 2016 (Part 1)*. Retrieved June 14, 2018, from http://www.legalexecutiveinstitute.com/artificial-intelligence-in-law-the-state-of-play-2016-part-1/

Ortiz, J. (2016, June 14). *What are the Requirements of a Cloud First Strategy?* Retrieved October 8, 2018, from https://storageswiss.com/2016/06/14/requirements-of-a-cloud-first-strategy/

Petrillo, M., & Baycroft, J. (2010). *Introduction to Manual Annotation.* Retrieved August 10, 2018, from https://gate.ac.uk/teamware/man-ann-intro.pdf

Poibeau, T., Saggion, H., Piskorski, J., Yangarber, R., & (eds). (2013). *Multi-source, Multilingual Information Extraction and Summarization.* Springer-Verlag Berlin Heidelberg. doi:10.1007/978-3-642-28569-1__2

Poole, C., & Prouse, R. (2017). *NUnit*. Retrieved October 8, 2018, from https://nunit.org/

Royce, W. (1970). Managing the Development of Large Software Systems. *Proceedings of the IEEE WESCON*, (pp. 1-9).

Russell, S. J., & Peter, N. (2010). *Artificial Intelligence: A Modern Approach* (3 ed.). Pearson Education, Inc.

Selenium. (2018). *What is Selenium?* Retrieved October 8, 2018, from https://www.seleniumhq.org/

Sharma, S., & Coyne, B. (2017). *DevOps For Dummies* (3rd IBM Limited Edition ed.). John Wiley & Sons, Inc.

SmartBear Software. (2018). *TestComplete*. Retrieved October 8, 2018, from https://smartbear.com/product/testcomplete/overview/

Tape, T. G. (n.d.). *The Area Under an ROC Curve*. (University of Nebraska Medical Center) Retrieved August 10, 2018, from http://gim.unmc.edu/dxtests/roc3.htm

The Research Group Agile Knowledge Engineering and Semantic Web (AKSW). (2017, January 21). *Precision, Recall and F1 measure*. Retrieved August 14, 2018, from https://github.com/dice-group/gerbil/wiki/Precision,-Recall-and-F1-measure

Thomas, B. (2018, February 12). *Security at the Speed of DevOps*. Retrieved July 27, 2018, from https://www.tripwire.com/state-of-security/devops/security-speed-devops/

Walber. (2014, November 2014). *Precision and recall*. Retrieved August 10, 2018, from https://commons.wikimedia.org/wiki/File:Precisionrecall.svg

Weyuker, E. J. (1982). On Testing Non-testable Programs. *Computer Journal vol.25 no.4*, (pp. 465-470).