TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

PEKKA PENNANEN

UTILISING CLOUD COMPUTING IN SOFTWARE TESTING

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 1st March 2017

ABSTRACT

PEKKA PENNANEN: Utilising Cloud Computing in Software Testing
Tampere University of Technology
Master of Science thesis, 58 pages, 0 Appendix pages
May 2018
Master's Degree Programme in Information Technology
Major: Software Engineering
Examiner: Prof. Hannu-Matti Järvinen
Keywords: software testing, test automation, cloud computing

Cloud computing is a relatively new and growing industry [1]. In this thesis, the goal is to find how cloud computing can be used for software testing. Software testing is essential for software engineering. Its purpose is to create information about the quality of a software product [2]. Another goal of this thesis is to start utilising cloud computing in software testing at M-Files company.

Literature was reviewed for information on software testing, cloud computing, and how the latter can be used to perform the former. After the literature review, a case study was conducted with the goal to utilise cloud computing in M-Files' software testing. The purpose was to increase M-Files' test automation capacity. In the case study, a method of how to migrate M-Files' automated integration testing to the cloud was identified. Then the migration was executed according to the identified method. The case study is described in detail in this thesis.

The literature review found that software testing in the cloud has many benefits, such as potentially significant cost savings. The cloud was also found to have issues, such as data location regulations which restrict its use. In the case study, M-Files' automated integration testing was migrated to the cloud. The migration was done by using Teamcity to launch virtual machines in Microsoft Azure. An already existing NUnit test set was used in the cloud. As a result of the migration project, up to 90% of tests have been run using the new cloud setup. The time it takes for test automation to complete was cut by 27% from 12.8 to 9.3 hours. The migration project accumulated an upfront cost of 3,588 euros. In addition to the initial cost, during a 30 day follow up period, an operational cost of 160.52 euros was accumulated. The costs were deemed acceptable.

The migration project enabled M-Files to increase the number of builds that are automatically tested. As the project's result, test automation in the cloud has become an integral part of testing in M-Files' R&D department.

TIIVISTELMÄ

PEKKA PENNANEN: Pilvilaskennan hyödyntäminen ohjelmistotestauksessa
Tampereen teknillinen yliopisto
Diplomityö, 58 sivua, 0 liitesivua
Toukokuu 2018
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Ohjelmistotuotanto
Tarkastaja: Prof. Hannu-Matti Järvinen
Avainsanat: ohjelmistotestaus, testiautomaatio, pilvilaskenta

Pilvilaskenta on verrattain uusi ja kasvava teollisuudenala [1]. Tässä työssä päämääränä on selvittää, kuinka pilvilaskentaa voidaan hyödyntää ohjelmistotestauksessa. Ohjelmistotestaus on välttämätön osa ohjelmistojen kehitystä. Sen tarkoituksena on tuottaa tietoa ohjelmiston laadusta [2]. Tällä työllä on myös tavoitteena alkaa hyödyntää pilvilaskentaa M-Files nimisen yrityksen ohjelmistotestauksessa.

Työssä tehtiin kirjallisuuskatsaus, jonka tavoitteena oli kerätä tietoa ohjelmistotestauksesta, pilvilaskennasta ja miten pilvilaskentaa voi hyödyntää ohjelmistotestauksessa. Kirjallisuuskatsauksen jälkeen toteutettiin tapaustutkimus. Tapaustutkimuksen tavoitteena oli hyödyntää pilvilaskentaa M-Filesin testiautomaatiossa. Tarkoituksena oli kasvattaa M-Filesin testiautomaatiokapasiteettia. Tapaustutkimuksessa määritettiin keino siirtää M-Filesin automatisoitu integraatiotestaus pilveen, minkä jälkeen siirto toteutettiin. Tapaustutkimus esitetään yksityiskohtaisesti tässä työssä.

Kirjallisuuskatsauksessa selvisi, että ohjelmistotestauksella pilvessä on monia hyötyjä, kuten rahalliset säästöt. Pilven käyttöön liittyy myös ongelmia, kuten tiedon sijaintiin kohdistuva sääntely. Tapaustutkimuksessa M-Filesin automatisoitu integraatiotestaus siirrettiin pilveen. Siirto toteutettiin käyttämällä Teamcityä, joka käynnistää Microsoft Azureen virtualisoituja tietokoneita. M-Filesillä jo käytössä olevaa NUnit testikokoelmaa käytettiin myös pilvessä. Siirron lopputuloksena jopa 90 prosenttia testeistä ajetaan pilvessä. Aika, jonka yksi testikokoelman suorittaminen vaatii, laski 27 prosenttia eli 12,8 tunnista 9,3 tuntiin. Siirto kerrytti 3588 euron etukäteiskustannuksen. Lisäksi 30 päivän seuranta-ajan aikana kertyi 160,52 euroa käyttökuluja. Kulujen määrä todettiin hyväksyttäväksi.

Testauksen siirto pilveen on mahdollistanut M-Filesin kasvattaa automaattisesti testattavien ohjelmistoversioiden määrää. Projektin lopputuloksena pilvessä testaamisesta on tullut olennainen osa M-Filesin kehitysosaston toimintaa.

PREFACE

I would like to thank Tero Piirainen and Minna Vallius for giving me this thesis subject. I thank Tero for reviewing the thesis and all support I've received. I also want to thank all my colleagues in M-Files who have helped to make this thesis. Also, thanks to Professor Hannu-Matti Järvinen for the guidance I received.

Last but not least, praise to my family and friends who have supported my studies.

Tampere, 13.5.2018

Pekka Pennanen

CONTENTS

FIGURES

TABLES

LIST OF ABBREVIATIONS AND SYMBOLS

API	Application programming interface (API) is a set of methods that enable communication between software components.
AWS	Amazon Web Services (AWS) is a subsidiary of Amazon.com that provides cloud computing resources.
CD	Continuous Delivery (CD) is a software engineering approach in which teams work in short cycles, ensuring that the software is in a condition where it can be released at any time.
CI	Continuous Integration (CI) is a software development method in which development team members integrate their work frequently.
DNS	Domain Name System (DNS) is a naming system for computers, services, or other resources connected to a network.
HDD	Hard disk drive (HDD) is a data storage device which stores and retrieves data using magnetic storage.
IaaS	Infrastructure as a service (IaaS) is a model in which a service provider offers infrastructure components, which are hosted in the cloud.
ISV	Independent software vendor (ISV) is an organisation that makes and sells software.
PaaS	Platform as a service (PaaS) is a model in which a cloud service provider offers an externally controlled platform for a customer's systems and software.
R&D	Research and development (R&D) refers to activities undertaken by organisations to develop or improve services or products.
SaaS	Software as a service (SaaS) is a model in which a cloud service provider offers readily setup applications.
SQL	Structured Query Language (SQL) is a language used in programming and designed for managing data held in relational database management systems.
SSD	Solid-state drive (SSD) is a data storage device that uses integrated circuit assemblies to store and retrieve data.
SSH	Secure Shell (SSH) is a protocol used for secure data communications.
TaaS	Testing as a service (TaaS) is a model in which software testing is outsourced to a third party.
UI	User interface (UI) is the space where human-computer interaction occurs.
VM	Virtual machine (VM) is an emulation of a computer system.

VPN	Virtual private network (VPN) extends a private network across a public network.

1. INTRODUCTION

Cloud computing is a relatively new and growing industry that is gaining more traction in software business [1]. New ways to utilise this resource are constantly developed. In this thesis, a look will be taken into one field that can utilise cloud computing: software testing.

Software testing is essential for software engineering. Its purpose is to create information about the quality of a software product [2]. Testing evaluates the features of an application and discovers differences between the requirements placed for the software and the existing condition of the software [3].

Software testing based on cloud computing can be divided into two aspects. First, it can be software testing that is performed on an environment which resides in the cloud. Second, it can be testing the quality of cloud. [4] In this thesis, only the first definition is considered.

The goal of this study is to find how cloud computing can be used for software testing. It also has the goal of taking the gathered information into use and start utilising cloud computing in the software testing that the company M-Files does.

First, this thesis examines software testing, cloud computing, and how the latter can be used to perform the former. The examination is done in the form of a literature review. Then the thesis proceeds to a case study that aims to utilise cloud computing in software testing. The case study was done for the company M-Files. In the case study, a method of how to migrate M-Files' test automation to the cloud was identified. Then a project to do the migration was executed based on the identified method. The case study's planning, execution, and results are described in detail in this thesis.

The organisation of this thesis is as follows. Chapter 2 describes software testing and agile software development. Chapter 3 defines cloud computing and discusses its aspects. Chapter 4 describes utilising cloud computing in software testing, benefits and issues of it and how to transition into using the cloud. In Chapter 5, a detailed case study is given. The case study shows a description of transitioning a software

company's testing to the cloud. Chapter 6 evaluates the results of the case study and provides discussion about future development options. Chapter 7 concludes the thesis with a summary of the thesis.

2. SOFTWARE TESTING IN AGILE DEVELOPMENT

Software testing is the process that aims to provide information about the quality of a software product [5]. In practice this means that a software tester tries to find bugs in the software [6]. The number of bugs found compared to the number of test cases run offers information about the quality.

This chapter discusses agile software development and how software testing fits into it.

2.1 Agile software development

Multiple different agile software development methods have been developed in order to answer to the need for developing functional software for customers in a fast manner and to withstand changes to software requirements during the development process. Many of the methods are rooted in the Manifesto for Agile Software Development [7]. The manifesto states that:

> *We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:*
>
> - *Individuals and interactions over processes and tools*
> - *Working software over comprehensive documentation*
> - *Customer collaboration over contract negotiation*
> - *Responding to change over following a plan*
>
> *That is, while there is value in the items on the right, we value the items on the left more.*

The manifesto conveys that for a more successful software project a more agile procedure is necessary. Cooperation with the customer and a working software are more

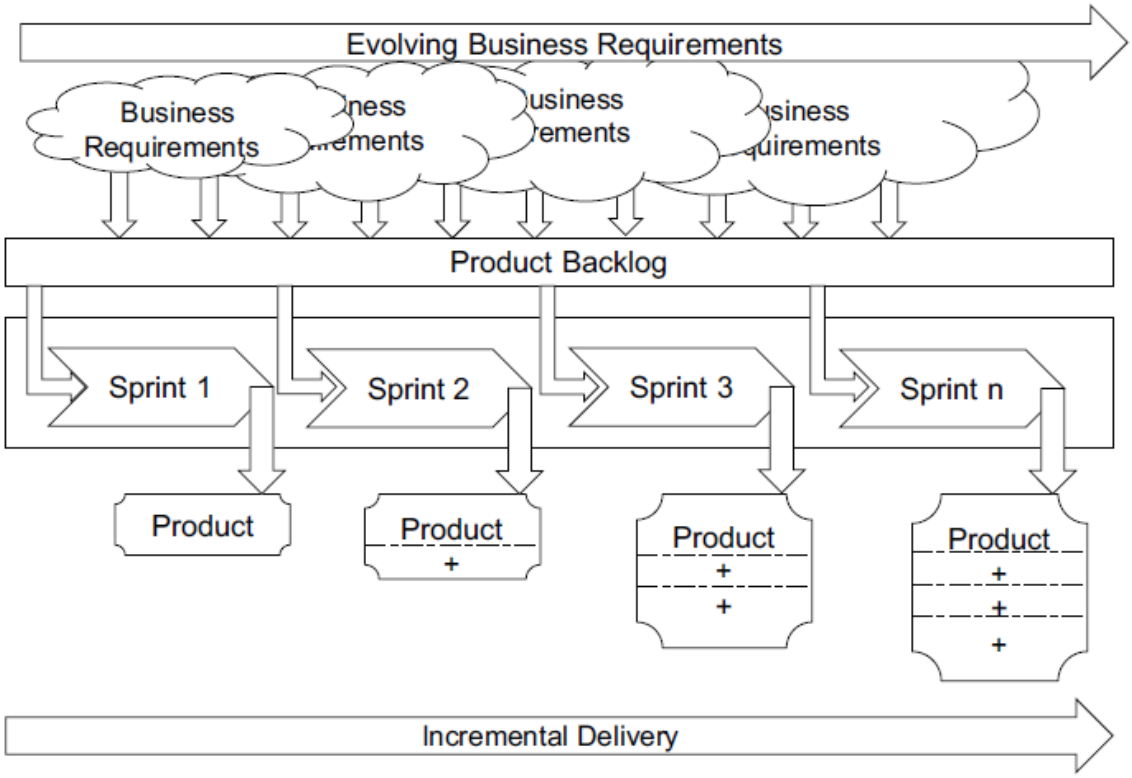

Figure 2.1 *Agile (Scrum) project life cycle. [2]*

important than exhaustive documentation and contract negotiations. The point is to be ready for change instead of strictly following premade plans and processes.

There are many agile development methods such as Extreme Programming, Scrum, Crystal, Kanban, and Feature-Driven Development. They all follow the principles expressed in the Agile Manifesto. Figure 2.1 describes an example of a project life cycle in agile development in the case of Scrum. A project that follows Scrum method is composed of many iterations which are called sprints. Each sprint usually results in new functionality that is ready to be delivered to customers. The added functionality can be either new functionality or enhancements to existing functionality. Sprints normally last from one week to four weeks. Normally, the number of sprints in a project is not defined at the start because in agile projects the exact requirements are not fully understood at the beginning of a project. Customer requirements, which are usually called user stories, are added to a product backlog. The customer requirements evolve during the project. Product is delivered incrementally as the result of sprints. [2]

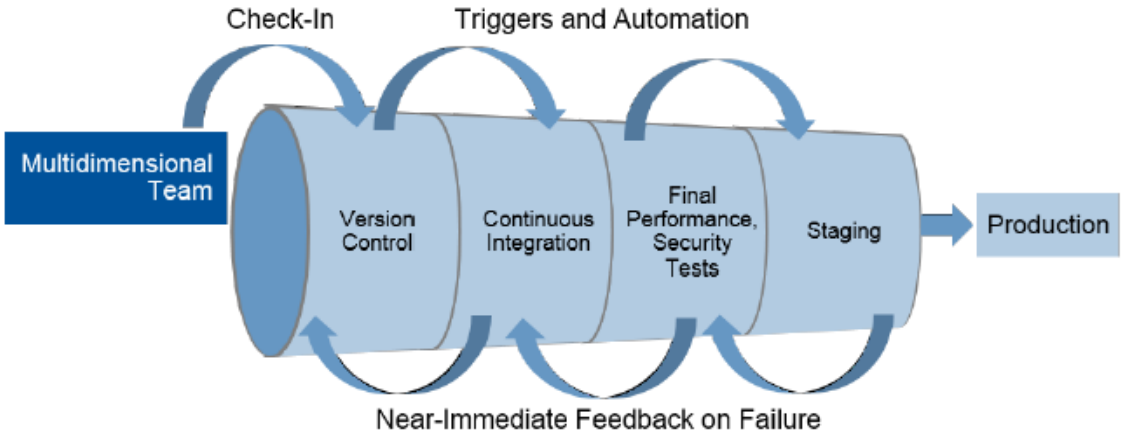

Figure 2.2 *CI/CD process. [8]*

2.2 Continuous integration and delivery

Continuous integration (CI) is a software development method in which development team members integrate their work frequently. Typically, each team member integrates their work at least once per day which leads to multiple daily integrations. Each integration is then automatically built and tested in order to find possible errors as soon as possible. This practice reduces integration problems and allows a more rapid development of a software application. [9]

Continuous delivery (CD) is closely related. It is a software engineering approach where teams produce software in short cycles and ensure that the software is at any time in a condition where it can be reliably released. Continuous delivery should let companies to quickly, efficiently, and reliably bring improvements to their services to market. Eventually this would allow staying ahead of competition. [10]

Figure 2.2 depicts a CI/CD process. An automated process takes developer check-ins through the CI/CD pipeline and feedback is near-immediate.

2.3 Software testing

This section will cover what software testing is, why it is necessary, and how it is done. Also software testing in agile software development is covered.

2.3.1 Definition, purpose, and execution of testing

Software testing is the process of inspecting a software to find differences between the requirements placed on the software and the existing condition of the software, the difference being bugs. Testing also aims to evaluate the features of the software. To put it short, software testing is the process of analysing or executing a program with the intention of discovering bugs. [3]

The main goal of software testing is to provide information about the quality of the tested software. It also aims to find information about the residual risk of how much of the software remains untested. Ultimately, this mitigates software stakeholders' risk of poor product quality. [2]

Defects appear to a person using a software product because of errors in the application. A defect that does not surface while using an application does not have any impact on the software. Only when a defect occurs under the correct conditions, it can cause the software to fail. A software failure may have serious consequences such as compromised reputation of a business, endangered business or user safety or viability of a business. [2]

It is not possible to create a software application without errors. Thus software testing is an imperative task to be done before shipping to customers. The goal is to minimise the possible errors in the software product. Additionally, information about the software quality is necessary for company decision makers. Software applications do not always do what is expected of them, so they need to be verified and validated. Software testing also needs to be done throughout the life cycle of a piece of software and its development. [2]

There are six main principles to software testing that a software engineer should understand in order to conduct effective testing [3]. The principles are the following:

- Tests need to be traceable to requirements. The purpose of this is to discover any software errors that may cause the software to fail to meet the client's requirements.
- Tests should be planned long beforehand test execution. After the software requirements have been gathered, test planning can begin.
- The Pareto principle is valid for software testing. This means that 80 percent of bugs discovered during testing are likely to be caused by 20 percent of all software components. Thus these problematic components should be detected and thoroughly tested.

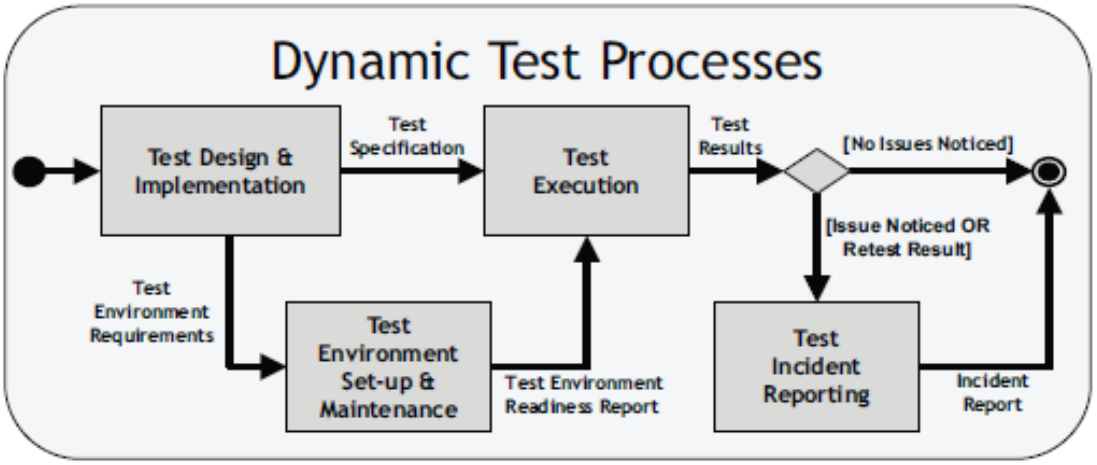

Figure 2.3 *Dynamic test processes. [2]*

- Testing should progress in a bottom-up manner. Testing should begin with planning and executing tests on individual components. Later on testing should move its focus to integrated clusters of components and lastly the entire system.
- It is impossible for testing to cover everything. Every combination of paths is impossible to test due to the large number of permutations. However, it is possible to cover the software in an adequate manner.
- For effective testing, the tester should be an independent third party. The software engineer who created a software item is not the best person to test it. This is due to the developer being driven by delivery whereas an independent tester is driven by quality.

A major challenge in software testing is to determine what parts of the software to test and which can be left untested. Resources are not limitless and thus a major challenge, time pressure, drives what can be tested. [5] In addition to executing tests, designing and implementing test cases takes time. Also setting up and maintaining test environments and composing issue reports are time consuming tasks. These tasks are phases of dynamic test processes and they are presented in Figure 2.3.

Software testing is done in five phases. Testing begins during the construction of a piece of software, during which testing is done individually to each software component. Once the components have been combined to sub-systems, the testing effort enters integration testing phase. When sub-systems have been integrated together the system testing phase begins. Finally, when the software product is released, the

testing enters the maintenance phase. In this phase, the software keeps changing due to bug fixes and additions of new functionality. All changes to the software are necessary to be tested to ensure that the software quality remains on an acceptable level. [5]

2.3.2 Agile testing

In agile software development, testing does not wait for software components to be ready before testing begins. Instead, testing contributes to the development effort throughout the development cycle. [11]

See Figure 2.4 for a comparison between traditional and agile testing. In a traditional scenario, an application is planned and developed to completion and only then tested at the end of a project, right before shipping. Contrarily, agile testing needs to take into account agile development methods which means developing the software in small parts. Each small increment of coding is tested after being finished. A single iteration might be as short as a week. [11]

In agile testing, testing is not done based on requirements gathered at the beginning of a project. Rather, tests need to be done based on the requirements of each story. Moreover, agile testing also aims to ensure business value and delivering quality to customers instead of the traditional viewpoint of only meeting requirements. In agile testing, even if a story passes the tests that were made based on its requirements, more testing is done to better understand the requirements and how the feature should work. A story can only be said to be done if all testing tasks are finished. [11]

2.4 Test automation

This section covers the topic of test automation. The task of performing manual testing can take a great deal of time and effort. To amend this, software test automation can be utilised.

There are many ways to conduct test automation. This section focuses on the three layers of test automation presented in the test automation pyramid, which is illustrated in Figure 2.5. It describes the foundation of test automation to be unit tests. The foundation represents the bulk of testing which supports all other layers of testing. The second layer of tests is the integration layer. These tests operate at the Application programming interface (API) level. [12] They are tests that verify

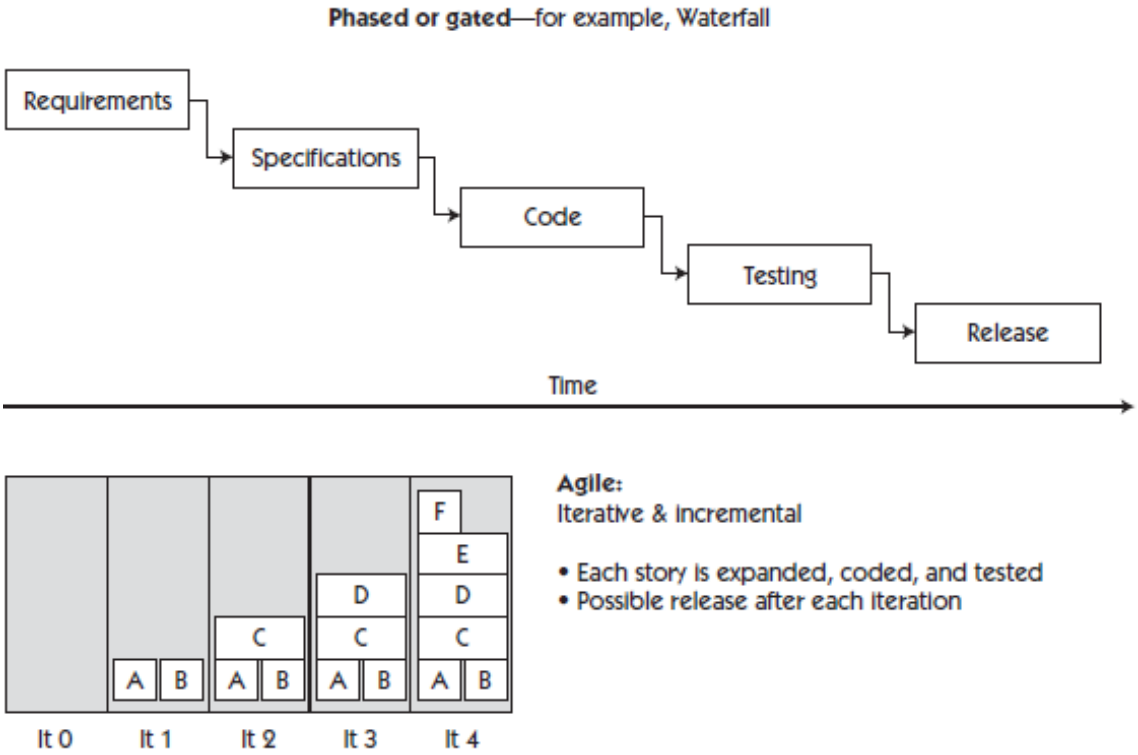

Figure 2.4 *Traditional waterfall testing and agile testing. [11]*

functionality directly without using a user interface (UI) [11]. The top layer represents UI tests which should be the smallest test effort. They are tests that are done using the UI. [12] Most systems also require manual testing to supplement test automation. This is shown as a cloud at the tip of the pyramid. [11] Manual testing will be looked into in Section 2.5.

2.4.1 Unit testing

Testing on the lowest level of a software is called unit testing or module testing [6]. This means testing software components individually to ensure their correct operation. The components are tested independently without any other system components. Unit testing focuses on verification effort. [3] Unit testing is usually tasked to programmers rather than a testing team [6].

There are many reasons to do unit testing. First of all, in unit testing it is easier to isolate bugs. When a bug is found in unit testing, the tester can be certain that the issue lies within the tested unit. Secondly, in unit testing, the tested module is small enough that it can be attempted to be tested in an exhaustible fashion. Thirdly, in unit testing, one can eliminate the risk of confusing the interactions of various different errors in separate parts of the software. [3]

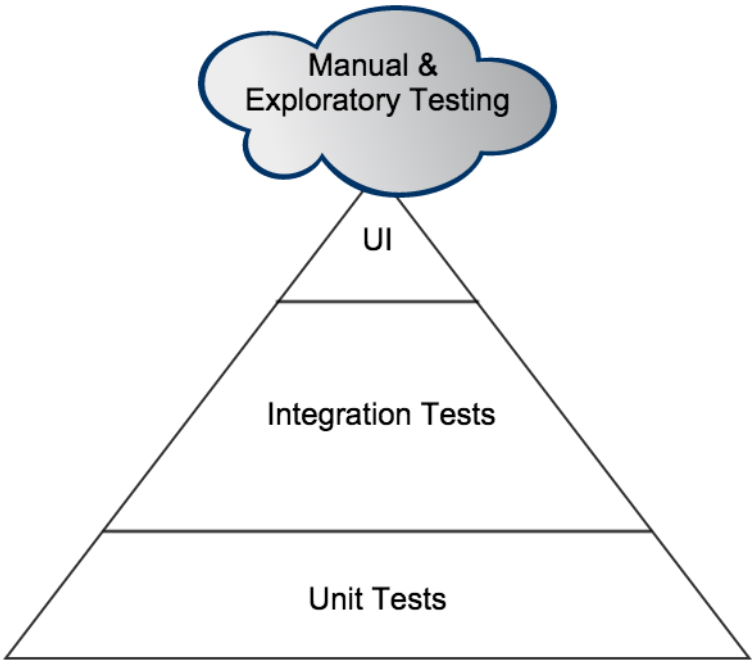

Figure 2.5 *Test automation pyramid. [12]*

There are many test frameworks that can be used for unit testing. For example, NUnit is a unit testing framework for all .NET languages. The framework contains a runner that can be used for automation of the unit tests. [13] NUnit is one of the options that Microsoft suggests to use for unit testing when developing software that uses their .NET framework [14].

2.4.2 Integration testing

When individual software components have been tested, they will be integrated and integration testing is performed against groups of modules [6]. Integration testing means testing that different software components work correctly together. The tests are designed and executed against APIs, Windows services, or any interfaces exposed between system components. Integration tests usually need the tested application to be installed or deployed in an environment similarly as it would be delivered in production. [12]

Often the UI of a software is based on an API. When testing is done in the integration layer, the variations and permutations of API calls are tested more efficiently and robustly than if the testing was done on the UI level. This provides a well tested basis upon which a much smaller set of UI tests can be built. [12]

Integration tests are often done using a unit test framework [12]. For example, xUnit tools such as NUnit and JUnit can be used to perform integration tests [11]. Integration testing can be tasked either to programmers or testers [12].

2.4.3 UI test automation

User interface is the main way, and for most users the only way, users interact with an application. Thus, user interface is an important part of software and should be tested.

Many of both large and small scale applications have usability issues that some groups of users face. In order to discover these issues, efficient and effective means are necessary. Automated usability or accessibility tests can provide information in software development process while the program is being developed. With this fast and visible feedback developers can quickly fix problems in the software. This also enables developers to experiment with greater confidence. UI test automation also helps discovering potential issues in internal releases by testing each release quickly and consistently. [15]

Layout problems are an example of UI issues that a user may encounter. They can have an adverse effect on a user's perception of an application. They may also reduce an application's usability by distracting or frustrating its users. Layout issues may be caused by e.g. localising an application from one language to another. Traditionally layout problem discovery has relied on human testers due to the challenge of finding these issues with test automation. This is no longer the case and now this is one field that can benefit from UI test automation. Some automation frameworks, such as WebDriver, offer the possibility to create layout tests. One way such tests are made possible is the detection of text on a web page and then comparing the location of said text to the detected location of text boxes and fields. If the text meets or overlaps the edges of a field, an annotated screenshot will be captured for further review by a human tester or developer. [15]

Automated usability tests can be valuable additions to manually performed tests. Automatic UI tests do not replace human testers but complements human testing. Effective test automation increases the overall value of testing by extending both its reach and range. The large mass of automated UI tests would be impractical to be done by human testers because of, e.g., the vast set of web pages that are tested. Contrariwise, tests done in person can spot many issues that automated tests struggle to detect. [15]

UI test automation can, in addition to valuable information, find irrelevant issues. This is a problem with test automation, as the bugs it finds may be unlikely to be seen by users, or developers do not see value in fixing them. [15]

For test automation to be good, it requires similar skills, practices, and passion as software development does. Many test automation tools require their users to be skilled in technical and programming matters in order to be able to write tests. Open source tools may be too difficult to use. Some test automation tools are simplified to enable more people to be able to write new tests, but this is often a bad trade-off. [15]

2.4.4 Concerns regarding test automation

Even though test automation is a powerful tool, it is not an answer to everything. Following realities regarding test automation are to be considered [6]:

- The software evolves constantly. Test automation needs continuous maintenance to account the changes made into the software.
- Automation does not substitute human testers. Test automation cannot achieve everything, some software issues are better noticed by humans.
- Verification is hard. One needs to make sure that automated tools can efficiently handle changes.
- One can easily place too much trust on automation. Even though test automation reports zero issues that does not mean there are no bugs to be found.
- One should not spend too much time on working on automated testing tools instead of testing the software.
- Test automation development should follow the same standards and guidelines that the tested software follows.
- Test automation tools can be invasive and cause software failures. A bug found by automation should be tried to be re-created by hand to find out if the tool is the cause of the problem.

Because test automation is not the be-all and end-all, it is usually supplemented with manual testing which will be looked into in the following section.

2.5 Manual testing

As mentioned above, even though test automation is a powerful tool it does not replace manual testing performed by humans. Rather, automated tests complement human testing. In practice, defect detection at a system level is largely dependent on manual testing effort of human testers. Most of new defects are discovered by manual testing. [16] Test automation removes the need to perform simple and repetitive testing tasks from humans and allows manual testers to use more time on creative testing [17].

Test documentation and planning is an important part of manual testing. The most important deliverables are a test plan, test cases, bug reports, metrics, statistics, and summaries. The test plan describes the method used to verify that the software meets its specification and customer needs. The plan includes at least quality objectives, resource needs, schedules, assignments, and methods. Test cases list the specific items to be tested and define detailed steps that are followed in testing. Bug reports depict issues found using the test cases. [6]

Exploratory testing is one type of manual testing. It is a testing technique in which a software tester based on their experience designs and executes test cases spontaneously. The tests have a basis on the tester's earlier knowledge, prior experience with the tested product, and heuristic methods to find common types of software failure. [2] Exploratory testing is an imperative part of testing in agile software development [11].

When exploratory testing is practised, typically test cases are not designed or documented in advance. The testing is based on the tester's own intuition, curiosity, and the results they got from previous tests. [2]

3. CLOUD COMPUTING

This chapter discusses cloud computing. It explains what cloud computing is and the different ways it can be set up. The chapter also gives a look into the most major companies providing cloud computing services.

3.1 What is cloud computing

National Institute of Standards and Technology NIST defines cloud computing as a model for an omnipresent, convenient, and on-demand pool of configurable computing resources accessible over a network. These resources can then be rapidly both taken into use and released. Using the resources requires minimal effort from both the customer and the provider. [18]

The NIST definition is only one of the cloud computing definitions as the concept is quite loosely defined. There are different explanations to what cloud computing is but the general acknowledgement is that it includes virtualised hardware, effectively unlimited storage, and necessary software for a client to access the infrastructure. [5]

3.2 Service models

Cloud computing is divided to three different service models, the models being Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). This section shortly covers the meaning of the three models. [18]

3.2.1 Infrastructure as a service

Infrastructure as a service is a model in which a service provider offers resources such as servers and storage capacity. The resources are scaled according to customer needs. Virtualisation allows the provider to share actual physical resources between multiple customers. Amazon EC2 is a well known example of this model. [5]

The services that the provider delivers are not limited to only hardware. Servers, storage, and network are accompanied by associated software such as an operating systems virtualisation technology and a file system. IaaS is an evolution of traditional hosting as IaaS does not require any set time of commitment from customers. Instead, customers are able to provision resources on demand. IaaS provider does not do much management apart from keeping their data centre operational. Users are to deploy and manage everything else by themselves. [19]

3.2.2 Platform as a service

Platform as a service is a model that offers an externally controlled platform for a customer's systems and software. The model typically includes an operating system, software development tools, databases, and a full infrastructure including servers to host the customer's applications. Microsoft Azure follows this service model. [5]

Platform as a service provides development and deployment applications without the cost and complexity of purchasing and maintaining the infrastructure underneath. All necessary facilities for developing and delivering applications and services are provided as a whole. A virtualised and clustered grid computing architecture is frequently the fundament for this model. Some PaaS providers offer a specific programming language or API, such as Google AppEngine which is a PaaS offering where developing is done in Python or Java. [19]

3.2.3 Software as a service

Software as a service is the simplest service model from a customer's point of view. In this model, the customer gains access to a readily setup application that is hosted in the cloud. The software is accessible online via a thin client interface such as a web browser. The customer does not manage the application apart from small user-specific configurations. SalesForce is an example of a service with this model. [5]

In software as a service model, the customer does not purchase the software in the traditional sense. Instead, the customer pays for what they use. Billing may be based on user count or some other kind of consumption basis. The service provider usually hosts and manages the application in their own data centres. Some providers choose to provide the software while hosting it on some other provider's IaaS or PaaS model service. [19]

3.3 Deployment models

In addition to the service models discussed in the Section 3.2, cloud services are divided by the way their infrastructure is arranged. There are four solutions for the infrastructure setup: public, private, hybrid, and community cloud. [18] Service models are orthogonal to deployment models [20].

3.3.1 Public cloud

A public cloud's infrastructure is owned by the service provider and it is hosted at the provider's premises. Customers have no control of the infrastructure which is shared by multiple customers. [5] A public cloud service provider has its own policy and value. The provider also has profit, costing, and charging models of their own.[20]

A public cloud is available for the general public's use. Its service provider may be a business, academic, or government organisation. Combined ownership between different kinds of organisations is possible. [18] Public cloud is the dominant form of cloud computing deployment. Many popular cloud services, such as Amazon EC2, follow this deployment model. [20]

3.3.2 Private cloud

A private cloud is accessible only to its owner and the owner's associates. The infrastructure is privately owned and managed. This solution offers a possibility for better customisation, standardisation, security, and privacy. The trade-off is that a private cloud can be more expensive than a public cloud. [5]

A private cloud can be set up within an organisation to maximise and optimise the utilisation of existing in-house resources. Data transfer cost from local IT infrastructure to a public cloud is fairly considerable and thus evading it with a private cloud may be preferable. A private cloud helps companies to have full control over their mission-critical activities. [20]

3.3.3 Hybrid cloud

A hybrid cloud is a combination of public and private solutions. In regular use, the organisation uses its own private cloud but when necessary, an overflowing peak in

usage is covered with public cloud capacity. This is to ensure that excess load does not jeopardise the service. [5]

Cloud-bursting is related to hybrid clouds. In this scenario, the most critical applications are hosted on a private cloud. Other not as security-sensitive applications are hosted on a public cloud. Amazon has an example of a hybrid solution called Virtual Private Cloud (VPC). [5]

3.3.4 Community cloud

A community cloud is solely used by organisations which all are part of a single community. The community cloud may be owned by one or more of the participating organisations or by a third party. [18] For example, government organisations may share cloud infrastructure for state-related cloud needs [5].

The cloud community formulates a degree of economic scalability and democratic equilibrium [20]. The organisations share common goals for the cloud, such as security or compliance considerations [18].

3.4 Cloud providers

It has become possible to offer cloud computing resources at prices that are competitive to traditional computing costs. This has enabled cloud providers to emerge as a new business area. The competitive pricing is possible because of the large scale of the data centres. When considering network, storage, and administrator costs, a data centre with approximately 50,000 servers is five to seven times relatively more inexpensive to build and maintain than a smaller data centre of 1,000 servers. [21]

In this section, a look is taken at some of the companies that provide cloud computing resources. Amazon and Microsoft are the leaders of cloud service market by a large margin followed by a multitude of smaller players. Figure 3.1 shows the June 2017 evaluation of cloud providers by Gartner. Since 2016, Amazon and Microsoft alone have accounted for almost all of infrastructure consumption in cloud business and most customers will choose either of them. Thus, this section will focus on the two market leaders. Many of the leaders' competitors face significant business challenges and their customers face notable supplier-related risks. [22]

Figure 3.1 *Gartner Magic Quadrant from June 2017 evaluates the major cloud providers. [22]*

3.4.1 Amazon

Amazon was the first company to start providing cloud computing capacity that customers could use to run their own software. The most notable Amazon cloud computing service, Amazon EC2, was announced in 2007 and it allowed anyone to purchase cloud computing resources for 0.085 dollars per computer-hour with no minimum or maximum purchase and no contract. [23] Amazon Web Services (AWS), a subsidiary of Amazon, has been the market share leader in cloud providers for over 10 years. The leader position has been held by AWS in the latest Gartner evaluation of cloud providers in June 2017. AWS offers XEN-virtualised single- and multitenant computing with multitenant storage and a large variety of additional services. The

AWS marketplace also offers a multitude of third-party software and services. [22]

AWS has data centres all around the globe, e.g., the U.S., Ireland, Australia, Singapore, and Brazil. AWS is also willing to negotiate large-scale single-tenant solutions, such as a dedicated U.S. federal government region. AWS strongly appeals to agility-oriented IT buyers but is also frequently chosen by safety and efficiency oriented IT buyers. [22]

Amazon's biggest strength is the dominant market leader status. AWS continues as the thought leader of the industry to which all other competitors are compared. Amazon continues to offer new innovations added to their already large portfolio of services. AWS is the safe choice in the market, being the most mature, enterprise-ready provider with the best capabilities of governing a large number of users and resources. [22]

On the downside, AWS's large portfolio requires expertise to take into use. It is easy to get started with AWS but optimal use, such as best practices and cost management, can be challenging to even expert IT organisations. AWS is also the cost leader of the market and a reference point for pricing but AWS is not interested in being the lowest-cost bidder in competitive tendering. AWS's pricing structure is so complex that a third-party cost management tool is highly recommended. [22]

3.4.2 Microsoft Azure

Microsoft Corporation is the leading developer of personal-computer software systems and applications. Microsoft also publishes books and multimedia titles, has a line of hybrid tablet computers and gaming systems, offers e-mail services etc. Microsoft has a global presence in both sales offices and research and development (R&D) sites. [24]

Microsoft also entered the cloud business as a competitor to AWS with the launch of Microsoft Azure virtual machines (VMs) in June 2012 and general availability in April 2013. Azure offers Hyper-V-virtualised multitenant computing with multitenant storage. Their service portfolio is also broadened by numerous IaaS and PaaS capabilities, such as object storage (Blob storage), Azure Container Service, and a batch computing service (Azure Batch). Azure marketplace also offers third-party software and services. [22]

Microsoft Azure is divided to two deployment models: classic and resource manager. They represent different ways of deploying and managing Azure solutions. They have

different API sets and deployed resources can contain major differences. The two models are incompatible with each other. Originally, classic was the only deployment model. In this model, all resources exist independently without any possible grouping of related resources. Azure introduced resource manager model in 2014. It added the concept of a resource group, which is a container for resources that share a common lifecycle. This offers certain benefits, such as the possibility to deploy, manage, and monitor a set of services as a group instead of handling them all individually. [25] Some features are not currently supported by the newer resource manager model. Migration is supported from the classic to the resource manager model.[26]

Just as AWS, Azure has data centres all around the globe, e.g. the U.S., the U.K., Korea, and Brazil. Azure also has six data centres dedicated to U.S. federal government, two of which are dedicated to the Department of Defence. Microsoft Azure appeals to both traditional safety and efficiency oriented IT and agile IT. Traditional IT customers value the ability to use Azure to extend their Microsoft relationship and investment in Microsoft technologies. Agile companies value Azure's ability to integrate with Microsoft's development tools and technologies. Agile companies might also be interested in Microsoft's integrated specialised PaaS capabilities, such as Azure Machine Learning. [22]

Microsoft Azure's strengths are its large market share, being second only to AWS. Azure has maintained a high growth rate with the estimated 2016 revenue being 3 billion US dollars. Microsoft continues to add new features to Azure with an accelerated velocity on top of the already very capable and broad platform. Microsoft is also adding new innovations instead of primarily copying competitor capabilities. Microsoft is able to bundle Azure with other of their products and services. Thus Azure is often chosen as a strategic cloud provider by customers that are already committed to Microsoft technologies. [22]

As a drawback, Microsoft Azure's service experience is not as enterprise-ready as expected from a company with such a long history as an enterprise vendor. Microsoft has issues with technical support, documentation, training, and breadth of the independent software vendor (ISV) partner ecosystem. These issues are being addressed by Microsoft and significant improvements have been made. Microsoft professional services' implementations have inconsistencies regarding quality. The issues make it challenging for customers to gain expertise and mitigate risks. As a result, customers have an increased reluctance to deploy production applications and migrate data centres to Azure. [22]

Also, while Microsoft keeps improving their capabilities in security, availability, per-

formance, networking, flexibility, and user management, the functionalities are not on the level that enterprise customers expect. Determining right implementations is hard due to Microsoft's multiple generations of solutions which are accompanied by unclear guidance. DevOps-oriented customers may be disappointed by the lack of Azure support in some open-source and third-party tools and software. [22]

4. UTILISING CLOUD COMPUTING IN TESTING

Software testing can be done with computing capacity residing in the cloud. An organisation debating moving software testing to a cloud environment can use SMART-T method to determine whether the transition is feasible. The SMART-T method is looked into in Section 4.4.

This chapter looks into how cloud computing can be utilised in software testing. The potential of the cloud will be studied from the perspective of test automation. Also the benefits and issues of the cloud will be addressed.

4.1 Test automation in the cloud

Software testing in the cloud changes the traditional testing scenario by utilising a cloud service provider's infrastructure to gain resources in order to reduce test execution time, increase test execution cycles available, and increase the efficacy of testing. The end goal is to improve the quality of the application being tested. [5] Constructing virtual test environments in the cloud is feasible due to the on-demand nature of the cloud. [4]

Test automation's benefits increase over time when test cases are used repeatedly. These benefits are increased when utilising the cloud due to the larger number of tests that can be run in cloud infrastructure compared to an on-premise solution. Thus, the importance of test automation increases when migrating to the cloud. [5]

All levels of testing can be performed using the cloud. If a software product is based on the cloud, testing it in the cloud can bring the test environment to be identical or close to the production environment. [27]

Software testing in the cloud can be divided into four patterns [4]:

1. Cloud testing in a private cloud platform. In this situation, software testing is carried out using a private cloud environment. All software testing is done

by the software organisation themselves. This is a non-outsourcing model with strong security.

2. Cloud testing in a public cloud. Software testing is done using a public cloud environment. The cloud environment can be the same as the company's internal environment, only moved to the cloud. While implementing the testing, a private cloud testing platform can be used. This model is also non-outsourcing and high security.

3. Cloud testing in a private cloud while outsourcing testing to a third party. In this model, a software organisation may be worried about the safety of source code. A good cooperation requires that both partners share a good relationship of trust.

4. Cloud testing is completely outsourced to a third party software testing agency. The third party testing agencies can use public cloud platforms for the testing. This model is the main mode of Testing as a Service (TaaS). TaaS means that testing is viewed as a service which can be completely supplied by a third party. Cloud computing can supply more than just the infrastructures or software through the internet but also the related service.

4.2 Benefits of using the cloud

Most customers buy cloud capacity in order to gain greater business agility or to access infrastructure capabilities that are not possible with data centres of their own. Cloud computing can also offer significant costs savings when customers have short-term, seasonal, disaster recovery, or batch-computing needs. Also cloud computing can be a great benefit for small companies or companies with limited funds that cannot afford investing in an on-premises infrastructure. Additionally, if an organisation suffers from non-efficiency regarding their own infrastructure, moving to the cloud along with streamlining and automating their operations increases the likelihood of achieving savings. [22]

The largest-scale cloud providers continue to lower their prices and automated managed services will substantially lower the cost of infrastructure management over time. As such, it is expected for cost advantages to continue accruing to the cloud providers. [22]

Using cloud services is secure enough for most workloads and customers. Most major cloud computing service providers offer a high degree of security on their platform.

Security is not only the service provider's responsibility, a customer also needs to do their part to have their controls configured correctly. [22]

The natural isolation between machines in cloud virtualisation can assure that a malfunction in one machine will not affect the rest of the system. This greatly increases the robustness of testing. Also dynamically allocating resources is convenient due to the possibility of having the virtual machine configurations adjusted by the system. [4]

4.3 Issues and risks of using the cloud

Customers do not always save money by using cloud computing. Many customers start investigating the option to use cloud computing to achieve cost savings, but in the end, most customers buy cloud computing capabilities for other reasons than savings. For large enterprises with internal data centres, well-managed virtualised infrastructure, efficient IT operations teams, and a high degree of automation, utilising the cloud for steady-state workloads may be more expensive than an internal private cloud. [22]

Data location is a concern for some customers. Usually customers prefer to have their data at the same region as their operations are in order to minimise network latency. However, there are also regulatory concerns which may require keeping data in a certain country. Additionally, there have been revelations about intelligence agencies obtaining access to private data, which has increased the desire of non-U.S.-based companies to purchase cloud services from local non-U.S. providers. However, local providers usually lack the scale and capabilities of the global providers and thus they may focus on providing services for small businesses. Furthermore, having the data stored locally does not guarantee avoiding domestic or foreign surveillance. [22]

4.4 Transition to the cloud

Transition of software testing systems from on-premises to the cloud can be a laborious project. The cloud is not always the better option of the two and thus the decision about migration should be carefully considered in organisations.

To ease the decision making about migrating testing to the cloud, a decision framework called SMART-T has been introduced. SMART-T consists of three parts: business drivers, technical factors, and operational results. Each part is dedicated to answer one of three key questions regarding migrating to the cloud. See Figure 4.1 which shows the detailed steps of SMART-T. [5]

The first part of SMART-T is the business drivers section. It answers the question 'why migrate testing to the cloud'. This question can be answered by investigating if cloud computing would be faster, more economic, or better compared to on-premises computing. The investigation should lead to an answer whether the migration is desirable from business point of view. [5]

The second part of the framework is the technical factors section. This part aims to answer the question 'when to migrate testing to the cloud'. Moving testing to the cloud is not always the best solution and it can be costly and laborious. Not all test cases are possible to be migrated to the cloud without changes. Some reengineering may be required to be done on test code, libraries, and dependencies. Also the cloud environment needs to be taken into account. The desirable characteristics of a cloud-based environment should be gathered and the availability of such cloud environments examined. These factors should be considered in order to find out if migration to the cloud is feasible at the moment. [5]

If transitioning to the cloud is seen as feasible in the current situation, then the last part of SMART-T begins which is the operational results section. It answers the question 'how to migrate testing to the cloud'. The section consists of a trial migration and if the trial's results are acceptable, an actual migration. The trial migration consists of three steps: a stakeholder workshop, a pilot study, and evaluating the initial results of the pilot study. The goal of the workshop is to agree on the migration project details. Then a pilot study is conducted. The pilot study is meant to be representative of the actual migration, but with reduced complexity. Once the pilot has been finished, its results are evaluated. [5]

If the results of the pilot study are acceptable, the last part of the operational results section begins, which is the actual migration. It consists of three steps. First, documenting the migration guidelines and overall process takes place. Then migration estimates are adjusted based on the pilot study results and a migration plan is formalised. Lastly, the actual migration is executed according to the migration plan. [5]

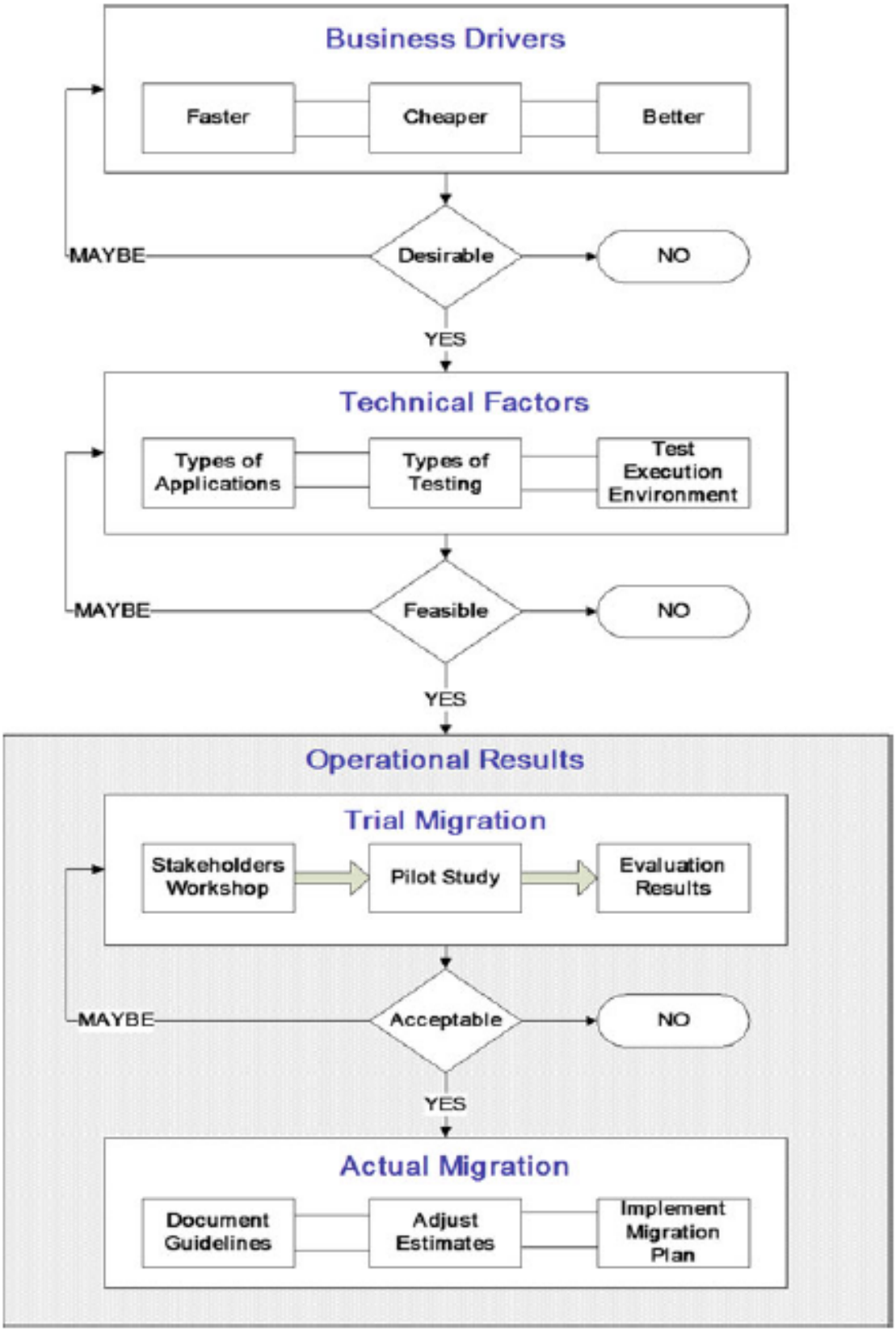

Figure 4.1 *SMART-T is a tool to help deciding whether migrating testing to the cloud is sensible. [5]*

5. CASES IN A SOFTWARE COMPANY

As a part of this thesis, a migration of software testing from on-premises computing capacity to the cloud was done for a software company. This chapter discusses what was done in this case. The next chapter continues on the topic of this case by evaluating the results.

5.1 M-Files

This section covers the company M-Files in which the case occurred. First the company itself and its product are discussed. After that a brief look on the current development and testing practices is done.

5.1.1 Company

M-Files is a Finnish software company that specialises in enterprise information management solutions. The company's headquarters is located in Hervanta, Tampere, Finland. Other company offices are located in the United States, the United Kingdom, France, Germany, Sweden, and Australia. [28]

M-Files was founded in 1988 and it employs approximately 400 personnel. The company moved to its current industry of enterprise content management in the year 2002. The company's revenue was 38.6 million euros in 2016. [29]

5.1.2 Product

The main product of M-Files is an enterprise content management software also called M-Files. Instead of a traditional folder based system, M-Files organises objects by their metadata. The important question is what a file is, not where it is saved. M-Files can be flexibly deployed either on-premises, in the cloud, or on a hybrid combination of the two. The cloud solution is called M-Files Cloud Vault and it is based on Microsoft Azure. [28]

M-Files software is used by thousands of customer organisations in over a hundred countries. Examples of well known Finnish customers are Nokian Renkaat, Patria, and R-Kioski. [28]

5.1.3 Current development and testing practices

M-Files' research and development department follows a practice of continuous integration. A continuous integration solution called Teamcity has been selected as the software that handles the test automation in the company. Teamcity is used to launch NUnit and web UI tests whenever a new build has been detected. The NUnit test set is the point of interest in this case. It consists of over 15,000 test cases that take on average 12.8 hours to run with current on-premises test machines. The test set performs integration tests on M-Files API.

In addition to test automation, every user story is manually tested by a quality assurance engineer. The engineer goes through the feature with its developer, designs a set of tests, and performs them. A user story can only be accepted as done once a quality assurance engineer has completed testing and found issues have been processed.

Before this migration project, test automation was driven solely in the company's own server capacity that was hosted on-premises. As time had passed, the existing on-premises capacity had become insufficient to meet the demand that test automation has. Especially during peak times, the existing capacity could become overworked with numerous builds waiting in queue. This was not an optimal situation from R&D point of view and thus needed to be solved.

The capacity problem was not possible to be solved by increasing the number of on-premises servers. This was firstly due to the fact that there were no more room for testing server machinery in the current office setup.

A second issue in increasing the on-premises capacity was that it was not viewed as a sustainable solution because the need for computing capacity is ever increasing. Even if the current spacing issue was to be solved, the same issue would be faced in the near future again. A third issue was that purchasing new server equipment is a large upfront cost and after the initial purchase, the machinery still accumulates costs due to the maintenance that it requires.

Additionally, it is beneficial to bring the test environment as close as possible to production environment. M-Files Cloud Vault is a cloud product so it would be beneficial to test it in a cloud environment, namely in Azure.

5.2 Available alternatives and considerations

To answer the problems discussed in Section 5.1.3, it was proposed that the additional computing capacity would be purchased from the cloud. This way there are no upfront costs and any additional machinery maintenance costs disappear. The only costs that happen are accumulated by the amount of usage that the cloud meets. However, in order to gain capacity from the cloud, numerous options needed to be considered. These options are discussed in this section.

5.2.1 Cloud arrangements

At first, the angle on how to utilise the cloud needed to be decided. In Section 4.1, it is stated that software testing in the cloud can be divided into four patterns. The patterns can be summarised as follows:

1. Testing in a private cloud platform. Testing is not outsourced.
2. Testing in a public cloud platform. Testing is not outsourced.
3. Testing in a private cloud platform. Testing is outsourced to a third party.
4. Testing is purchased as a service from a third party, which can use public cloud platforms. This is the main mode of TaaS.

As this project is done in-house, the patterns three and four containing outsourcing of testing can be disregarded in this case. The already existing testing practices have been satisfying the need for testing and there was no will to transfer the testing effort or managing the tests to a third party.

The company at the time had no interest in setting up a private cloud. The benefits of a private cloud, such as heightened security and privacy, were not something that this project requires. A private cloud has a higher cost than a public one and none of a private cloud's properties were a requirement for the project. Additionally, M-Files already uses public cloud services provided by Microsoft Azure. Due to this, the natural direction is pattern two: moving the internal test environment to a public cloud platform.

The next decision to consider was to choose the provider for the public cloud, the alternatives being Microsoft Azure, Amazon, or some smaller party. Microsoft Azure was the choice. This is due to the fact that as was previously stated, M-Files is

already a customer of Azure on other company needs. M-Files is a Gold Cloud Platform Partner for Microsoft [28]. There was no will to start a customership with another provider, so Azure was a natural choice for test automation as well. M-Files Cloud Vault product is hosted in Azure, so testing it there would be beneficial as it brings test and production environments closer together.

5.2.2 Machine setup in the cloud

Once the course and a platform had been settled on, there was a need to consider how the the machines should be set up in the cloud. There were two choices determined for the instance of Azure: Azure Automation or Teamcity Azure Plugin.

Azure Automation

Azure Automation is a service for automating management tasks in the Azure cloud. It is based on PowerShell Workflow. The tasks are done by creating what Microsoft calls runbooks. The runbook automatises cloud management by allowing running tasks in Azure. [30]

The possible runbook solution for the M-Files project would have been creating a runbook which starts a suitable number of Azure virtual machines during the early hours of the day. These machines would use a suitable virtual machine image which would have everything ready for running the test automation. Once they are ready they would automatically connect to Teamcity and be ready to be utilised for the test automation. At the end of the work day, the machines would automatically shut down once they are no longer utilised.

The runbook solution has both advantages and setbacks. The major advantage is that it would be made by the project team for their specific purpose and thus it would work predictably for its purpose. We could trust that the machines are there ready and waiting when the runbook so administers.

The disadvantages for this solution would be that the machines would end up with idle uptime. This would accumulate costs for nothing else than the purpose of being immediately usable once a build is ready. If builds end up in the cloud rarely, the cost of keeping idle machines might be unreasonable.

Another problem is deciding when to shut down the virtual machines after office hours. You cannot simply shut them down once the office is closed, there might

still be test runs ongoing or pending. A possible solution would be to try to get information from the machine whether it is running a test set. If it reports itself idle, then move forward to shut the machine down.

In the best case, the runbook solution still ends up with idle time run on the machines and new machines cannot be started on-demand: there are only as many machines as are started each morning. The number of machines could be tuned when experience is gained from running the system to mitigate the problem. However, there still would be peak times such as a new release coming out. This would mean a temporal increase in the number of builds that need testing. This would need temporary manual changes to the system to answer to the surge in demand.

Teamcity plugin

The already used continuous integration software Teamcity has added Azure cloud support as a feature. This feature enables the possibility for Teamcity to automatically launch test runs in the cloud. The number of machines launched in the cloud is limited by the Teamcity licence. Adding more machines to the test pool requires a fixed number of running instance licences to be bought and added to the Teamcity server.

The Teamcity Azure plugin is divided to two releases: classic plugin and resource manager plugin [31], [32]. The plugins respectively support the classic and resource manager deployment models of Azure. Refer to Section 3.4.2 for the difference between the two deployment models. Both plugins allow running Teamcity builds in the cloud with virtual machines that the plugin automatically launches. The classic plugin is in feature freeze state but is a more mature product with more options.

There are some differences between the features of the two plugin versions. The classic plugin allows two types of virtual machine deployment. The first option is to use already existing classic Azure virtual machines which Teamcity can start and stop. As many machines as testing needs have to be prepared for Teamcity. The machines need to be created individually and then configured to be ready to run tests automatically upon machine startup. The machines, once ready for use, are then configured to the plugin. Once a build enters the Teamcity build queue, Teamcity will start some machine from the virtual machine pool to run the tests on. Once the tests have been run and the virtual machine becomes idle, it will either take the next build from the queue or if the queue is empty, it will be stopped by Teamcity. The stopped machines are deallocated and thus do not incur any costs

while not in use.

The second option for the classic plugin is to create new virtual machines. The prerequisite is to create the desired target virtual machine and configure it to be ready for Teamcity's usage. Once the virtual machine is ready it will be used to create a generalised virtual machine image. This image will be saved in .vhd format to Azure Blob storage. With this image, the Teamcity plugin can automatically create new virtual machines on-demand and run the tests on them. The new machines will be copies of the prerequisite virtual machine. Once the test runs are ready and no further builds are in the queue, the virtual machine will be deleted by Teamcity.

The resource manager plugin used to lack the first option to use already existing virtual machines to start and stop them. The resource manager was only able to use stored images to create new machines. There is a later added extra option to stop virtual machines created by the plugin and start them on-demand [33]. This is meant to emulate the classic plugin's start and stop behaviour. However, it is not a true a start and stop feature because it can only be utilised with machines created by the plugin. At the time, the project personnel were not aware of this largely undocumented, apart from a single Git issue, feature and could not consider if it would be beneficial. Although this feature came to knowledge later, it was then tested and found to be too broken to be suitable for any kind of use. See Section 5.4 for more information about the feature.

In addition to the emulating feature, Teamcity much later added a true feature of start and stop to the resource manager plugin as the "use existing virtual machine" option [34]. This is a new feature that was added after the project had been executed. Refer to the Section 6.4 for a suggestion to try this feature.

Considering Azure Automation and Teamcity

Azure Automation would be a simple solution, but the difficulties regarding automatic utilisation of cloud resources is a major setback. Idle uptime in the cloud is not an optimal situation. This problem is easily mitigated by Teamcity. Teamcity has no apparent drawbacks compared to Azure Automation so the decision was to go forward with Teamcity.

Next, the different Teamcity plugin options had to be considered. Both plugin options have their pros and cons. The classic plugin is the faster option due to it being able to start and stop existing virtual machines. Starting existing virtual machines is much faster that to create new virtual machines each time. The already existing

virtual machines are also simple to maintain. An administrator can start a virtual machine and use remote desktop connection to gain quick access the machine to do maintenance.

On the downside, with the classic plugin, maintaining multiple virtual machines does not scale. The administrator has to individually maintain all of the machines which can be very burdensome if there is a large number of virtual machines used by Teamcity. Any new machines added to Teamcity's pool also have to be created individually or the administrator can create one and make copies of it. It is much more time consuming to add new machines manually than using automation to create them on-demand. Thus, it is more difficult to react to increases in testing demand when machines are created manually.

The issue with the resource manager plugin is that the only option is to always create new virtual machines based on virtual machine templates. This is a much slower practice than to start already existing machines. The fact that starting a machine is slower was obvious and later measurements were made to confirm the exact time difference it takes between the two options. The Table 5.1 shows these measurements by comparing the time it takes to start running tests between the two options.

Table 5.1 *Time it takes from a build entering Teamcity queue to tests starting to run.*

	Step	minutes	total
Creating a new VM	Create VM	7	59
	VM startup script	8	
	Update Teamcity plugin	9	
	Fetch Git	35	
Starting an existing VM	Startup	4	43
	Update Teamcity plugin	4	
	Fetch Git	35	

As Table 5.1 shows, the resource manager plugin suffers greatly from having only the option to create new machines. The flaw is mitigated by the fact that this happens only for the first build that the virtual machine runs. If there are builds in Teamcity's queue the next build will have none of this delay as everything will be left ready by the previous build. Only the correct Git branch needs to be updated to the virtual machine but this is a minor operation taking only a maximum of a few minutes.

The option to create new machines also has its benefits. It enables the administrator to maintain a huge number of test platforms by only maintaining the template that

is used to create all the virtual machines. Maintaining the template requires more work than updating a single already existing virtual machine but if the number of virtual machines created from the template is large, the trade-off will be well worth it.

The template creation and update procedure is such that at first, a new virtual machine needs to be created. The machine will be configured to be ready for being utilised by Teamcity. Once the virtual machine is ready, it will be generalised and used to capture a template to Azure Blob storage. Teamcity can then use this template to generate new virtual machines. The capturing is done with PowerShell. When the template needs to be updated, a virtual machine instance of the template needs to be launched. This instance is then maintained as any other virtual machine would be. Then the updated instance is captured in the same manner as the first time. The newly captured image needs to be updated to Teamcity's configuration and it will be ready to use. It is a good practice to run a test build on the new image to see that it continues to function properly.

There are possibilities to mitigate the slowness of the resource manager plugin. See the Section 6.4 for some consideration. If the virtual machine creation time can be shortened, the resource manager plugin is the superior plugin choice as it brings the best of both worlds. Additionally, the classic plugin is no longer getting any new features.

A large benefit the classic plugin has is that for the initial part of the project, it is much easier to test the cloud machines when they can be made once and kept for long periods. Testing their functioning with quickly starting and stopping the same machine for multiple iterations is enormously faster than creating a new VM image for each time a machine configuration needs to be tried.

In the end, due to the lack of start and stop feature in the resource manager plugin, the classic plugin was chosen to be tested at first. It was deemed that the slow start time for testing in the resource manager version was too slow compared to the classic plugin. Additionally, the initial scale of the project was so small, consisting of two planned virtual machines, that maintaining them individually was of no concern. Also, it would be easier to make the initial system with the classic plugin because it is faster to develop when the machines can be tested without creating new images for every time something changes. After the initial phase, moving from classic to resource manager deployment model would be easy, if necessary in the future as Azure supports migrating from classic to the resource manager model [26].

5.2.3 Connecting cloud machines to the company network

Using Teamcity to launch the cloud testing machines had a problem regarding connecting Teamcity to Azure. This is due to Teamcity being hosted inside a firewalled company network. Two options were determined for solving the connection problem: either whitelisting the cloud machines' IP addresses in the firewall or using a virtual private network (VPN) to ensure access from the cloud.

The first option was deemed problematic. First of all, the problem with dynamically created Azure virtual machines is that they are dynamically assigned an IP address. This would render whitelisting impossible due to the ever changing IP addresses of the machines.

It would be possible to use ready-made cloud machines with purchased static IP addresses but this also would lead to more problems. Firstly, this decision would limit the possibilities of the project due to being forced to rely on only static handmade machines and abandoning the option of dynamic machine creation on need basis. A second issue would be that Teamcity only supports starting and stopping already existing virtual machines when using classic Azure deployments. This is the old way of using Azure so it would prevent moving to the new resource manager deployment model if such would be wanted. A third problem would be that using a static IP would also require whitelisting this external IP into the company network. This leads to some security concerns such as can we trust to always be the only ones in possession of this Azure IP. A benefit of the IP whitelisting solution would be that it does not require any external software or configuration apart from configurations in the firewall.

The second option of using a VPN seemed to be preferable. A VPN extends the company network to the cloud machines and as such no IP whitelisting of external IP addresses is necessary. The VPN also adds a layer of security to the transmissions between the cloud and on-premises. These points are clear benefits over using a static IP address solution. The downside of a VPN is that it can require an application to be installed in the machines that participate in the network. This is a minor issue because the software needs to be installed only once and then it can be left running. The VPN can also have a harmful effect on transmission speeds between the network nodes. The VPN also accumulates costs varying by the VPN provider.

Due to the numerous issues regarding whitelisting the virtual machines' IP addresses and the benefits of a VPN the latter option was chosen. M-Files IT-department offered Hamachi as the VPN solution. Hamachi is a VPN solution provided by the LogMeIn company. The service allows users to extend LAN-like networks to

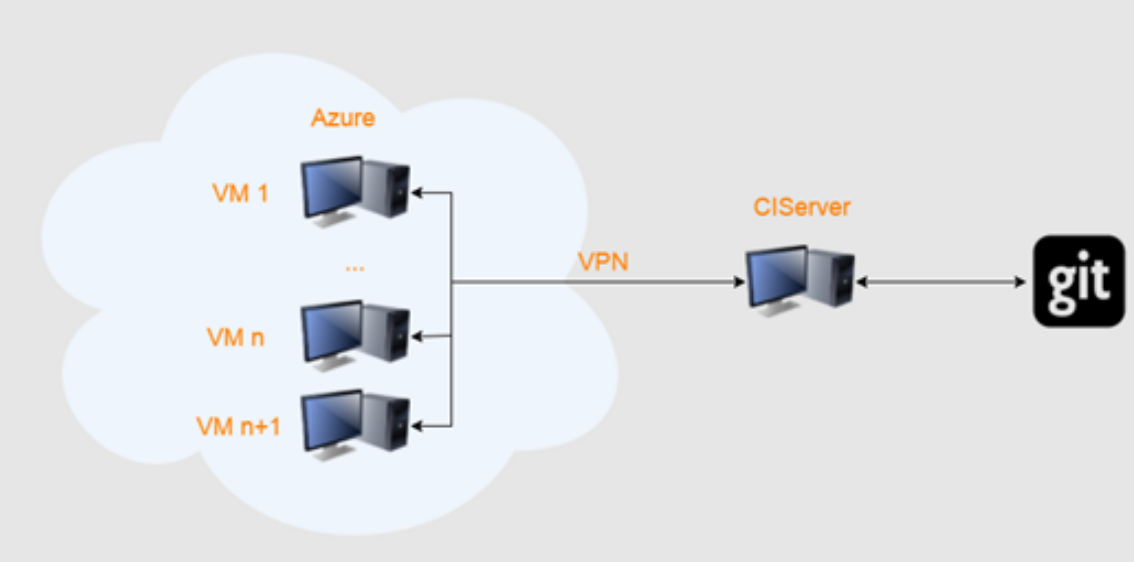

Figure 5.1 *The planned system setup. Hamachi is acting as the VPN between Teamcity CI server and the cloud while the CI server relays traffic between the cloud and the Git server.*

distributed sites on-demand. Hamachi incorporates a 256-bit AES encryption to the traffic creating a secure way of communication. [35] Hamachi seemed suitable and it was chosen to be tested with the project.

M-Files Git server is inaccessible from the cloud and has a very restricted nature. Due to this, installing Hamachi on the Git server was not an option. Therefore, it was decided that the Teamcity server was to be configured to work as a proxy server for Git. The proxy server would operate by relaying all Git related traffic between the Git-server and machines connected to the VPN. Figure 5.1 depicts the planned system setup.

5.2.4 Security

The aspect of security also needed to be considered during the project planning phase. All the major cloud providers have high security standards and have security audits performed on their services. [22] Microsoft Azure is no exception. Microsoft has taken extreme measures to provide security for their cloud customers. They have more than 20 cloud computing related security compliance certificates, among them ISO 27001 and 27018. [36]

Microsoft Azure has been considered secure enough for M-Files usage in other company matters, including production services aimed towards customers [28]. With some discussion among project stakeholders it was decided that Azure can be trusted

with M-Files software testing as well.

Of course, Microsoft alone cannot provide all the security needed. The highest risk regarding cloud services is not in the cloud, it lies internally within customer companies. Customer companies need to do their own part in being responsible with their systems in order to gain the best security. [36] Thus some actions were planned for the intended cloud virtual machines.

As discussed in Section 5.2.3, a VPN service was decided to be taken into use. It adds an additional layer of security to the data communications in the system which occur over the public internet. To bring further security, Teamcity can be configured to send all its traffic using HTTPS protocol. This will be configured into use.

Additionally for networking security, Azure has the option to define endpoints, which define the rules to permit incoming traffic to Azure virtual machines. The planned system would have only two endpoints enabled: one for Hamachi and one for remote desktop connection, which would be restricted to only company network IP addresses. Virtual machine's own Windows Firewall would also be set to block any other traffic apart from the two endpoint using connections.

Another security aspect to consider was the fact of how to protect company source codes. The used NUnit test set is built in a way that running the tests requires full M-Files source code to be present. Even though the Azure virtual machines were already deemed secure, as an extra precaution it was decided that the source code should only be put into an encrypted drive. The goal was to ensure that no-one else, including Microsoft, could not access the contents of the drive during use or after it is released from the use of M-Files. Azure shares its physical resources among its customers through virtualisation and with this solution data security could be ensured without trusting anyone else. Microsoft offers a disk encryption feature BitLocker with its Windows products which can be used to perform XTS-AES 128 or 256 encryption on disks [37]. This was deemed a suitable solution.

5.2.5 Azure virtual machine series

Once the cloud provider had been settled on, it was necessary to decide what type of a virtual machine would be suitable for the task. Microsoft offers a wide selection of different virtual machine setups in their catalogue [38]. The current on-premises machines have been well up to the task of running the test set so it was decided that at first, the cloud machines should resemble them in terms of performance, memory, and disk space. Different machine setups could be experimented upon later

to determine the best choice once a demo environment has been proved functional. The experimentation was performed later in the project and the results can be referred in Table 5.2 in Section 5.4.

Size D2v2 with Windows Server 2016 was chosen for the machine. D2v2 was the most similar to the machines that currently run the test automation on-premises. A D2v2 machine is based on a 2.4 GHz Intel Xeon E5-2673 v3 processor and has two CPU cores, 7.00 GiB of RAM, 100 GiB of temporary hard disk storage, and costs 0.227 euros per hour [38]. During this phase the D2-5v2 series machines were on discount which brought the actual cost down to 0.172 euros per hour [39].

The standard virtual machine series in Azure offers a permanent OS-disk and a temporary additional disk. The OS-disk is a slower hard disk drive (HDD) based option and the temporary disk is a faster solid-state drive (SSD) based option [38]. The temporary drive only retains its data during the time the virtual machine it is attached to is operating. When a virtual machine is restarted or shut down, the temporary drive might get erased of data. In order to gain the most of the machines, the testing task was decided to be done on the faster temporary disk.

5.2.6 Virtual machine startup preparation

In order to achieve encryption and opening the encryption in a fully automated system where machines start and stop without supervision, automation was required to perform the encryption task. Also some other tasks were necessary to be done upon machine startup. The automation was necessary to be designed to work reliably without any human monitoring. The goal was that when the automation is added to a VM virtual hard disk template, new machines created from the template will be completely automatically configured and taken into use. In order to achieve its goal, the automation should do the following:

- Either enable Bitlocker encryption or if already encrypted, unlock the encryption.
- Set used DNS servers as the company DNS servers. This is necessary to be incorporated into the script because predefined DNS configuration is lost when virtual machines are created from a template.
- Check if SQL users that the M-Files NUnit test set requires are present in the installed SQL server. If users are not present, they need to be created. This also needs to be in the script because the identity of a virtual machine

is different in a machine created from a template compared to the template machine. Users created in a template machine will not work.

- Git connection check. Teamcity is unreliable when connecting to a Git server for the first time. Teamcity will immediately fail the build being tested if connection to Git fails on the first time, which in experience it does quite often. This problem can be eliminated by doing a Test-NetConnection to see if Git is available for connection. If it is not available after a few repeats there is something wrong and automation should stop.

- Start Teamcity. Teamcity should not be allowed to start automatically during Windows startup because of the steps that are prerequisites for it to function. Thus, starting Teamcity should be handled by the automation task instead.

- Set SSH keys that the usage of M-Files Git requires. The keys need to be placed into the C:\Users folder of the user that runs Teamcity. Because the user identity is changed after new machines are created from a template, this needs to be done during startup. Experience showed that C:\Users folder of the user can take a surprisingly long time to be created by Windows during initial startup in a fresh Azure machine. This can be mitigated by doing this step as the very last task in the automation. Otherwise the script will either fail or additional waiting needs to be introduced to slow the script.

A Microsoft PowerShell script was designed to do the task. The script is automatically run on every machine startup. Figure 5.2 describes the process flow diagram of the script.

The script was to be incorporated to every cloud virtual machine for them to run it during each startup. The script will then fully automatically prepare the machine to be functional without any needed supervision.

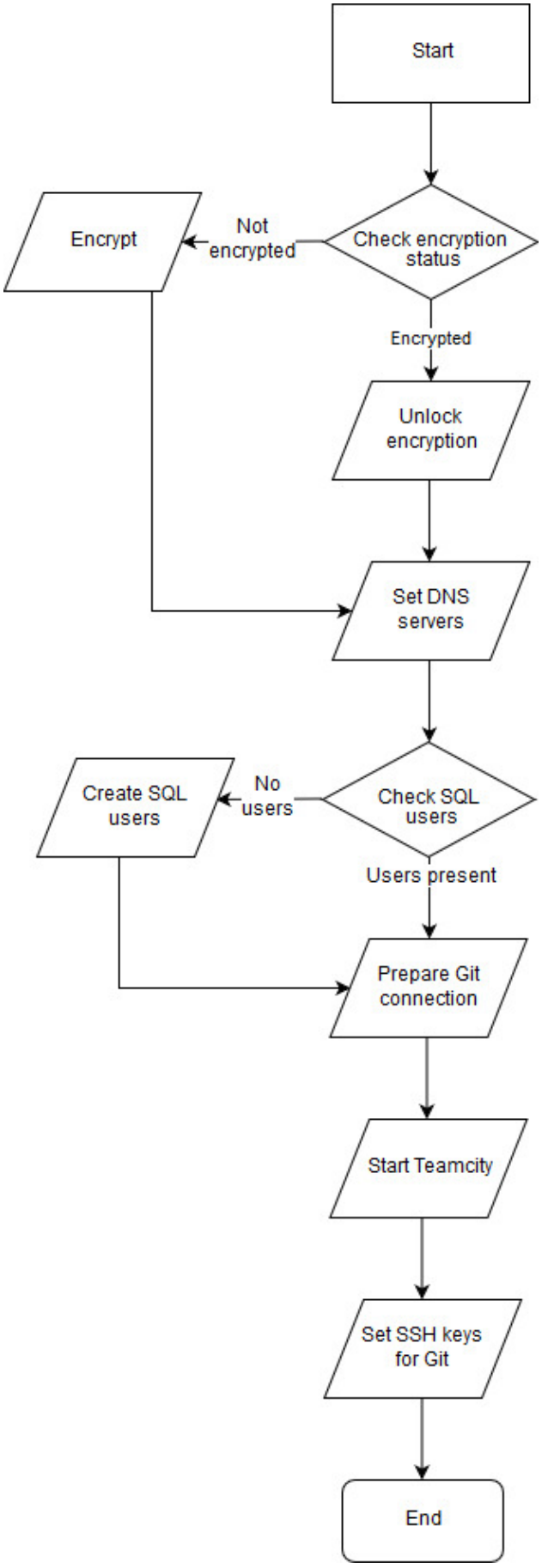

Figure 5.2 *Process flow diagram of the cloud virtual machine setup script.*

5.3 Project execution

Once the planning was done and different options had been considered and decided upon, the project entered a phase where the plan was to be executed. One virtual machine was created in Microsoft Azure cloud for testing purposes. The goal was to test the planned implementation on this machine. Once the machine would have been deemed ready for actual use, it would have been used as a template to make two production versions of the machine.

As described in Section 5.2.5, Azure D2v2 tier was chosen for the project. As per the plan, a D2v2 machine was created to the cloud. Once the virtual machine was deployed, the current test environment was duplicated there by hand. This was due to the fact that the on-premises testing machines were based on Windows Server 2008 R2 and Windows Server 2012. Copying them to the cloud was not productive. A Windows Server 2016 test machine was missing from the available on-premises test machines so creating one to the cloud was a good opportunity to add this platform to the test grid.

In addition to the duplicated test environment resources, such as a installation of Microsoft SQL server, Hamachi was installed on the machine. A Hamachi network was created and its client was installed to both the Teamcity server machine and to the Azure virtual machine. Also the startup script introduced in Section 5.2.6 was set to launch during each time a machine is started.

In practice Hamachi was found to be problematic. It has two working modes: relayed and direct. Direct is the fast and preferred option. As is suggested by its name, in the direct mode, the network traffic is sent directly between the VPN nodes. The other option, relayed, is used when direct mode is not possible. This may be caused e.g. by a firewall blocking direct communication between the nodes. The relayed mode redirects the VPN traffic through a relay server provided by Hamachi. [35] This is a much slower option than direct would be. Our network was discovered to be stuck in the relayed mode, causing the average transmission speed between a cloud virtual machine and Teamcity server to be a measured 734 kbit/s (0.09175 MB/s) in on-premises to cloud direction. This rendered the virtual machine useless due to the extreme time taken to update its Teamcity plugins and fetch source codes from Git. It would have required several days to start a test run.

There were attempts to resolve what was causing the VPN to remain in relayed mode. Nothing was discovered and colleagues more experienced with Hamachi stated that it often has this kind of issues with little ways to fix it. There were also additional issues discovered in Hamachi. Hamachi requires a VPN application installed on

all network nodes. In theory, this should not be a problem but practice proved otherwise. The application consumes large quantities of memory over time. This is not a problem for the short lived cloud machines but the always online Teamcity server started running out of memory after every week of uptime. This lead to a situation where the Hamachi process had to be killed and restarted weekly to prevent the server from running out of memory. It was also discovered that opposite to what was expected, the Hamachi application needs a specific installation package for each network node. This means that two virtual machines created from the same .vhd template cannot both connect to the VPN at the same time, otherwise if tried, one of the two nodes gets removed from the network. This forced to create a single template for each machine and prevented making multiple machines from one template. This drastically reduced the intended flexibility of the system.

While trying to find a solution to enable the Hamachi direct mode, also a search for other options was started with the intent of moving to some other VPN solution. Some colleagues recommended Azure Site-to-Site VPN as a solution that has been a positive experience in the past. Site-to-Site was deemed a suitable replacement. It is native to Azure and no VPN clients would be needed on any machines. Instead, Site-to-Site is separately configured to Azure by creating and configuring necessary VPN resources. The company end of the VPN needs to be configured into their firewall. The firewall product that M-Files uses was among the solutions supported by Azure for Site-to-Site. Site-to-Site seemed a suitable replacement for Hamachi and the initiative was taken further. The matter was discussed with company IT-department which agreed that Azure Site-to-Site was a desirable solution to the problem. The IT-department made the arrangements and configured both the Azure end and the company end to work with Site-to-Site. The Site-to-Site also provided direct access from Azure to the company Git server, which allowed scrapping the previous proxy server setup where the Teamcity server acted between the cloud and Git. This was a welcome simplification to the system.

A new virtual machine that has Site-to-Site enabled was created to test the new solution. Site-to-Site was an immediate success and Hamachi was scrapped. Site-to-Site achieved the measured average speed of 13.27 MB/s between the Teamcity server and a cloud machine in on-premises to cloud direction. After the network had been tested as functional, all software and configurations needed to run NUnit were installed in the virtual machine in order to try running the NUnit test set. The first results were encouraging: the VPN worked well and NUnit ran with only minor cloud related failures. These failures were addressed in the test cases to make them more robust in a cloud environment. The result was that now test runs were executed with the same pass rate in both on-premises and the cloud.

5.4 Fine tuning and moving to resource manager deployment model

With the Site-to-Site change the Azure deployment was also changed to resource manager deployment model. This decision was made to avoid needing to configure the Site-to-Site again in the future if the classic deployment model is deprecated at some point.

Resource manager model required a change in the virtual machine philosophy as the start-stop feature supported by the classic Teamcity plugin was dropped by the new resource manager plugin. Instead, in the newer plugin the goal is to automatically create new virtual machines in Azure based on virtual machine templates. In regards to this, Azure works as such that a virtual machine is needed to be created in the subscription and then the virtual machine is to be generalised and created to a .vhd template. The original virtual machine is practically destroyed in the process but now new virtual machines can be created using the template. The resulting machines are perfect copies of the original machine used for the template.

Table 5.2 *Azure virtual machine tiers, measured average time to run M-Files NUnit and operational costs by the hour and for a single test run.*

VM tier	NUnit duration	€/h	Test run cost (€)	Notes
A3	1640 min (27.3 h)	0.274	7.48	very low-speed
D2v2	556 min (9.3 h)	0.227	2.11	low cost
D2v2 promotion	*556 min (9.3 h)*	*0.172*	*1.60*	*discounted price*
D4v3	660 min (11.0 h)	0.358	3.94	low-speed
F4	662 min (11.0 h)	0.347	3.82	low-speed
H8	474 min (7.9 h)	1.439	11.37	fast and expensive

In order to learn which would be most cost effective, a set of different Azure virtual machines that would be suitable for this project's purpose were set up for testing. Each machine was tested for three times to measure the average time it takes to run the NUnit test set. Table 5.2 presents the measured average operational times and costs of the different system setups. Costs were derived from the Azure pricing page [38]. D2v2 virtual machine size has been discounted until further notice by Microsoft [39]. The discount has lasted the whole duration of the project so it was included in the table. As the table shows, the D2v2 machine is the most inexpensive of the tested options. It is also reasonably fast in running M-Files NUnit test set, being only second to the fastest H8 machine. H8 is highly costly in comparison but the run times with H8 are 14.7% faster than the second fastest D2v2. In the purpose of this project, the D2v2 machine was considered as the optimal middle ground between

costs and performance. The test automation machines were left configured at D2v2 choice which was the machine tier that was selected at first during project planning.

To ensure a more diverse testing environment, two template virtual machines were prepared, one with Windows Server 2016 and SQL server 2008 and other with the same Windows and SQL server 2016. Other relevant versions are already addressed by the on-premises test machines. The two machines were converted into virtual machine templates and were set to be the basis for one virtual machine per template so a total of two VMs was able to be created when needed. The system was limited to two Azure virtual machines in addition to the two on-premises machines at a time due to the installed Teamcity licenses that were limiting the machine number to four. The two licenses for cloud machines had been determined to be enough for the then current testing demand and more could be rapidly added if necessary.

The Teamcity resource manager plugin had after its release gained a feature that adds the possibility to automatically stop virtual machines created by it and later start them on-demand instead of always deleting created machines when they go idle and then creating new ones. This feature emulates the classic plugin start and stop behaviour but it still requires a VM image for an initial automatic VM creation. A hand-made VM cannot be utilised by this feature, the plugin needs to create the VM by itself. After the resource manager plugin was taken into use, this feature was decided to be tested.

Unfortunately, the preservation of VMs functionality was found to be too broken for usage. Teamcity would randomly create new virtual machines instead of starting the stopped ones. This would result in an ever increasing number of virtual machines accumulating in the Azure subscription which adds unwanted clutter and in the end might fill the subscription limit for the allowed virtual machine count. As a result, using this option would need regular check-ups and clearing the accumulated virtual machines.

In addition to the previous issue with the resource manager plugin feature that preserves virtual machines, the plugin had problems with stopping the virtual machines. From time to time, the plugin would only stop the virtual machines but not deallocate them. This is not acceptable due to stopped but allocated machines still incurring same costs as a running virtual machine. As a result with these issues, the resource manager plugin's option to preserve virtual machines was discarded as useless for our purposes for the time being. The initial thought of creating and deleting VMs according to demand was far superior.

The cloud deployment has since been successfully left running automatically. When-

ever the on-premises testing capacity is unable to answer to the demand, a cloud machine is created to remedy the situation.

Afterwards M-Files' R&D changed the test automation procedure so that all developer made builds are tested by test automation. Previously, only stable branch was tested for every build. This caused a large surge in demand for test automation capacity and new Teamcity licenses were added. Now the test automation grid in regards of NUnit consists of two on-premises machines and 12 licenses available for cloud machines. The same two template virtual machines are used for all of them, a maximum of six machines per template.

The cloud utilisation can now be seen as an integral part of the M-Files testing system. In a 30 day follow up period, a total of 90% of stable builds were tested using cloud machines.

5.5 Considering the project in regard to SMART-T

SMART-T, which was discussed in Section 4.4, was utilised as the workflow that guides the project. Business drivers for the project were obvious: testing would be faster if builds do not need to queue for a free test machine. Additionally, the testing would be better because M-Files is also a cloud product but had not been automatically tested on a cloud platform. The testing platform would be more close to production platforms. The testing could also be argued to be cheaper in the cloud due to no upfront machinery and real estate costs which would be considerable if on-premises capacity was to be increased. As a result, transitioning to the cloud was deemed desirable.

For technical factors, duplicating the current test automation setup on a cloud platform was deemed not a problem. The application and the testing to be done are suitable for a cloud environment so the transition to the cloud was seen as feasible.

A trial migration was initiated. Stakeholders discussed on how to execute the task and the framework for a pilot study was agreed upon. Pilot study's encouraging results can be seen in Section 5.4. The pilot had acceptable results and testing started to have a major tilt towards the cloud, resulting in 90% of the builds being tested on a cloud machine. This should be considered as the actual migration as the goal never was to have 100% share for cloud testing. Documentation concerning the project was created and from SMART-T point of view the migration can be considered as completed.

5.6 Manual testing using the cloud

During the project, in addition to test automation, also manual testing started to further utilise cloud computing. M-Files quality assurance team had previously had shared cloud platforms that hosted M-Files Cloud Vaults. Testers could configure a client connection to these vaults and so do manual testing with a cloud-based back end. The shared cloud platform was a good solution because all testing was done in a shared build to which all user story changes were merged. All user stories needed to have at least some testing with a cloud platform but the emphasis was with the on-premises installation of the product.

Testing procedure changed during the project and all user stories were to be tested in their own user story code branches. This made sharing the cloud vaults difficult and thus all testers were instructed to create a cloud platform of their own. This was accompanied with further encouragement to test user story features with a larger emphasis on the cloud than before. This has transferred the quality assurance team's manual testing effort to have a much heavier tilt towards cloud utilisation.

This works well with the notion of Section 2.5 that test automation only complements manual testing efforts performed by human testers. As a result, both manual and automatic tests are now utilising the cloud at a considerable emphasis.

6. EVALUATION AND FUTURE DEVELOPMENT

In this chapter, the case study of M-Files is evaluated for its results. The benefits and issues regarding the solution are discussed and a cost analysis is provided. Also possibilities are provided on how to further develop the system.

6.1 Benefits

The project has allowed a large increase in concurrent test sets that can be run without any delay. Up to 90% of builds have been run using the new cloud setup. The number of builds that are tested has increased considerably during the project and the number of builds that are tested in the cloud would be unsustainable to be tested on-premises. All of this is achieved with a low cost, see Section 6.3 for a cost analysis.

The project has also lowered the time it takes to do a test run from 12.8 to 9.3 hours with the selected D2v2 machines compared to the on-premises machines. This means a 27% decrease in test set completion time. This is a major improvement in performance and a large benefit provided by transitioning to the cloud.

An additional benefit comes from that now M-Files software is regularly automatically tested in the cloud. M-Files Cloud Vault is an Azure based product. Thus it is desirable and beneficial to bring the test platform as close as possible to the production platform.

6.2 Issues

The largest issue the project has is a slow startup time of virtual machines and slowness in initiating the first test run on each machine. As can be seen in Table 5.1, it takes 59 minutes to start test execution on a freshly created virtual machine. The biggest part of the problem is the large Git repository which is required to be

downloaded before tests can be started. It takes 35 minutes to download. The issue concerns only the first test run that is done on a virtual machine because fetching new Git branches takes only seconds and at most few minutes once the repository is present in the machine. Section 6.4 gives some thoughts about how this issue could be mitigated in the future.

A more general issue regarding the project is the increased maintenance the test automation requires. This is a natural cause of bringing more environments to the test system to be maintained. The issue is minimal as automation independently takes care of the machines. Windows updates and changes in the test environment are the only instances of required manual maintenance.

Another issue is the costs that the system keeps accumulating for as long as it is in use. This is also a minor issue due to the fact that the costs are very reasonable. Further cost analysis is available in the following Section 6.3.

6.3 Cost analysis

The project has three direct sources of costs: Azure virtual machines, networking costs, and Teamcity licenses. The virtual machine hourly cost per machine is 0.172 euros for the chosen D2v2 machine type [38]. Uptime of the virtual machines is unpredictable due to differing amounts of testing required at different times. However, after a 30 day follow up period with four cloud machines in use, it can be seen that they were actively testing for a total of 750.96 hours. This comes to a total cost of 129.17 euros. There is some minor overhead that should be added to this because each machine is configured to wait for 30 minutes for new builds to test before shutting down.

Networking costs are generated by the usage of the Azure Site-to-Site VPN. The costs consist of a basic type gateway and IP address lease. A basic type gateway costs an hourly 0.04 euros which comes to 28.8 euros for a 30 day period [40]. The gateway is always enabled, so it accumulates the full cost. Dynamic IP addresses, which the machines use, cost an hourly 0.0034 euros [41]. A 30 day lease for an IP would then cost 2.4 euros. Static IP addresses are more costly and unnecessary in this implementation. With 750.96 hours of testing during 30 days, the IP address lease comes to 2.55 euros. Thus the total cost of the VPN is the combination of gateway and IP costs which is 31.35 euros.

One Teamcity license costs 299 euros [42]. M-Files currently has 12 licenses available, which brings the total license cost to 3,588 euros. Because this cost is paid upfront

unlike the Azure costs which are charged based on usage, this cost is the most major part of the project expenses.

In addition to the initial license cost, Teamcity has a yearly upkeep cost of 149 euros per license. The upkeep includes support and Teamcity upgrades. The upkeep is not a requirement for using Teamcity. The current version remains functional even if the subscription expires. The upkeep needs to be paid if either support or upgrades are necessary. The initial license purchase comes with one year of subscription. [42] No upkeep licenses have been bought during the project because the initial subscription year of the licenses has covered the duration of the project.

Table 6.1 *Project costs*

Expense	Upfront cost (€)	Hourly cost (€/h)	30 day operational cost (€)
Azure VMs	-	0.172	approximately 129.17
VPN Gateway	-	0.04	28.8
IP Addresses	-	0.0034	2.55
Teamcity licenses	3,588	-	-
TOTAL	3,588	-	160.52

The project costs, both the upfront and the accumulating operational costs, are miniscule for a company of the size of M-Files. The costs are gathered in Table 6.1. Obviously, the costs are something that needs to be followed as time passes but as of now they are acceptable.

6.4 Future development possibilities

This section covers what future development options have been considered for the project. The options mostly try to improve testing performance. Also Azure Service Fabric is looked into.

6.4.1 Premium disks

The current setup uses standard HDD disks for the OS disk of the virtual machines and an SSD temporary disk for the Teamcity's work directory. The disks are the default options for Azure virtual machines of this price tier. The SSD disk is a temporary drive that is included in all Azure virtual machines. All data on the temporary drive will be lost upon virtual machine shutdown. [43] Due to this, the SQL database used in the tests is installed on the slow OS disk. It could be tested if using a more expensive premium SSD disk for the OS disk would improve the time it

takes to run the test sets. In addition to the OS, the SQL database used in the tests could perform better with a faster disk. Compared to the OS, especially the database has potential of improving the overall performance of the test set completion.

The possible improvement is very hard to estimate beforehand. The easiest option would be to create a test virtual machine and run some test sets in it as a trial.

6.4.2 Permanent disk

Azure has the option of creating additional permanent disks that can be attached to virtual machines. The disks can also be detached from virtual machines with ease. The disks can be either premium SSD-based models or standard HDD-based disks. [44] Using this kind of disk for Teamcity's work directory instead of the default temporary disk would greatly reduce the startup time of tests. This is due to the fact that these disks would no longer be formatted upon virtual machine shutdown. This practice would eliminate the need to fetch the whole repository upon machine startup or creation.

Another possible benefit gained from permanent disks would be that the Microsoft SQL Server used in testing could have its database storage placed on a disk of its own or to the same disk with Teamcity's work directory. Microsoft recommends avoiding OS or temporary disks with SQL Server database storage or logging [45]. This practice could bring an improvement to performance.

There are two downsides to this setup. First of all the permanent disks generate additional costs for the system. A 128Gb disk sufficient for usage would cost 18.29 euros a month for an SSD-based disk and 4.97 euros a month for an HDD-based disk for each disk [46]. The costs are arguably very moderate. It would be necessary to measure the test run time difference between SSD and HDD solution to determine which should be used.

The second problem regarding the setup is attaching the disk to a virtual machine. In the case of an already existing virtual machine this is not a problem but in the case of an automatically created virtual machine it is not obvious how to do the attaching. It was considered if attaching the disk could be automatically done with PowerShell during machine startup sequence. This would require a finite number of disks prepared for the machines that would be attached based upon their availability. This could probably be done but the required effort does raise concerns if it is not worth the workload.

It is also unknown what kind of impact a permanent disk would have on performance.

There is a risk that moving from the temporary disk to a permanent disk, both standard and premium models, might make the NUnit test set slower to finish. This is due to Microsoft stating that a temporary disk might have a higher performance rate than persistent storage such as permanent disks. [43]

6.4.3 Azure Blob storage repository cache

Azure Blob storages enable storing any type of text or binary data in the cloud. It is a service for storing large amounts of unstructured data that can be accessed via HTTP or HTTPS. One common usage of Blob storage is to store files for distributed access. [47]

Utilising a Blob storage to cache the Git repository used for testing could be a way to remedy the long startup time that the virtual machines are suffering from. This could be achieved by creating a small low-cost virtual machine to Azure with the sole purpose of fetching the latest commits from Git and placing them to the Blob storage. This could be done, for example, as a daily automated task.

Once the repository contents are cached in the Azure Blob storage, they would be readily accessible by the test agent virtual machines. If a requested version is missing from the cache, it would be prompted by the requesting virtual machine to get it. With this system, the virtual machines would get the repository from the cache instead of the actual Git server. This has the potential to improve the long time it takes to fetch the repository during the virtual machine setup. The improvement depends on Azure infrastructure to be faster in the transfer than the Site-to-Site VPN that is now used as the transmit medium. The actual effectivity of this solution is unknown until it is tested in the project as information to evaluate the effect beforehand is unavailable. Azure does not state how fast a transmission in the same region between a Blob storage and a virtual machine would be.

The cache master virtual machine could run a script or a small application that handles the upkeep of the Blob cache. On the test automation virtual machines, the required implementation could be included in the startup PowerShell script.

Alternatively, the cache virtual machine could act as a Git server by itself. It would regularly get the latest changes from the main Git server. The agent virtual machines would then use the cache virtual machine like the main Git server. This might be the superior option, but it cannot be said for certain as the difference in transfer speed between virtual machine to virtual machine and Blob storage to virtual machine is unknown.

6.4.4 Return to using already existing virtual machines

In December 2017, Jetbrains added the option to start and stop already existing virtual machines to the resource manager version of the Teamcity plugin. The feature became part of their February 2018 release of the plugin. [34]

It should be tested if this feature could be successfully used in this project. It might be beneficial to have, e.g., two always ready virtual machines that the plugin can start when necessary. Any possible overflow could be handled by the automatically created new virtual machines. This kind of setup would reduce the time it takes for new test requests to proceed.

The already existing virtual machine solution still suffers from using the temporary disk for the Teamcity work directory because once the machine is stopped the disk will be formatted. Thus this feature could also be combined with the permanent disk possibility described in Section 6.4.2. With a permanent premium disk attached, the stopped virtual machine would be ready to almost immediately start testing once turned on because the working directory would be preserved on the premium disk.

6.4.5 Azure Service Fabric

Azure Service Fabric is an application platform that helps deploying microservice and Docker container based applications in Azure cloud, on-premises, or in other cloud provider's cloud [48]. M-Files Cloud Vault is moving from virtual machines hosted in Azure Cloud Services to Azure Service Fabric. M-Files server will be hosted in a container accompanied by numerous microservices.

Testing should take this into account and make plans to move test automation to a similar environment. To ease this, it should be investigated if the current virtual machines could be converted into Docker containers to provide a straightforward transition. Docker offers a tool called Image2Docker which takes virtual hard disk images and suggests a Dockerfile based on the image [49].

7. CONCLUSIONS

The goal of this thesis was to find how cloud computing could be used for software testing. It also had the goal of taking the gathered information into use and start utilising cloud computing in the software testing that the case company does. The literature review conducted in this study demonstrated that cloud computing is a suitable platform for test automation that can be utilised in many ways. Software testing in the cloud was found to have many benefits, such as potentially significant cost savings. The cloud was also found to have issues, such as data location regulations restricting its use.

After the literature review, a case study of migrating software testing to the cloud was done. The test automation done in M-Files was studied and the means to migrate the testing to the cloud was identified. The identified course was to use Teamcity to launch virtual machines in Microsoft Azure and run the already existing NUnit test set there. A migration of the test automation to the cloud was successfully executed with minor changes to the original plan.

As a result of the migration project, up to 90% of tests have been run using the new cloud setup. The time it takes for test automation to complete was cut by 27% from 12.8 to 9.3 hours with the selected cloud machines compared to the on-premises machines. Despite the high utilisation rate of the new system, the cost of the migration remained low. The migration project accumulated an upfront cost of 3,588 euros. In addition to the initial cost, during a 30 day follow up period, an operational cost of 160.52 euros was accumulated.

After the cloud migration project was done, test automation in the cloud became an integral part of testing in M-Files' R&D department. The amount of testing that is done in the cloud would be unsustainable to be done on-premises. An additional benefit comes from that after the project was completed, M-Files software is regularly automatically tested in the cloud. M-Files Cloud Vault is an Azure based product. Thus it is beneficial that the test platform is as close as possible to the production platform. It can be concluded that the goals of the thesis were met.

BIBLIOGRAPHY

[1] L. Columbus, Roundup Of Cloud Computing Forecasts, Forbes, 2017. Referenced on 14th April 2018. Available: https://www.forbes.com/sites/louiscolumbus/2017/04/29/roundup-of-cloud-computing-forecasts-2017/#5b7d6d8c31e8

[2] ISO/IEEE/IEC, SO/IEC/IEEE 29119-1: Software systems engineering – software testing – part 1, 2013.

[3] B. Agarwal, S. Tayal, M. Gupta, Software engineering and testing, Jones and Bartlett publishers, LLC, 2010.

[4] S. Liu, J. Tang, J. Chen, S. Duan, L. Li, Q. Kuang, M. Cai, D. Hu, Virtual Software Testing Service Based on Cloud Computing, Applied Mechanics and Materials, Vol. 529, pp 739-742, 2014.

[5] S. Tilley, T. Parveen, Software Testing in the Cloud, Springer Berlin Heidelberg, 2012.

[6] R. Patton, Software Testing, Sams Publishing, 2000.

[7] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R.C. Martin, S. Mellor, K. Schwaber, J. Sutherland, D. Thomas, Manifesto for Agile Software Development, 2001. Referenced on 28th August 2017. Available: http://agilemanifesto.org/.

[8] D. Scott, T. Murphy, N. Wilson, Key Benefits of Continuous Integration and Delivery Combined With Cloud Computing, Gartner, 2016.

[9] M. Fowler, Continuous Integration, 2006. Referenced on 28th August 2017. Available: http://www.martinfowler.com/articles/continuousIntegration.html.

[10] L. Chen, Continuous Delivery: Huge Benefits, but Challenges Too, IEEE Software, 32(2015)2, pp. 50-54.

[11] L. Crispin, J. Gregory, Agile Testing: A Practical Guide for Testers and Agile Teams, Pearson Education, 2009.

[12] S. Ashman, Layers of Test Automation, QA Matters, 28th December 2014. Available: http://qa-matters.com/

[13] NUnit, the website of NUnit. Referenced on 4th March 2018. Available: http://nunit.org/

[14] Unit Testing in .NET Core and .NET Standard, .NET Core Guide, Microsoft. Referenced on 4th March 2018. Available: https://docs.microsoft.com/en-us/dotnet/core/testing/

[15] J. Harty, Finding Usability bugs with Automated Tests, Acmqueue, 9(2011)1. Referenced on 26th February 2018. Available: http://queue.acm.org/detail.cfm?id=1925091

[16] S. Berner, R. Weber, R. Keller, Observations and Lessons Learned from Automated Testing, Proceedings of International Conference on Software Engineering, 2005, pp. 571-579.

[17] J. Itkonen, M. Mäntylä, C. Lassenius, How do testers do it? An exploratory study on manual testing practices, Third International Symposium on Empirical Software Engineering and Measurement, IEEE, 2009.

[18] P. Mell, T. Grance, The NIST Definition of Cloud Computing, US National Institute of Standards and Technology (NIST), 2011. Available: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

[19] S. Bhardwaj, L. Jain, S. Jain, Cloud Computing: a Study of Infrastructure As a Service (IAAS), International Journal of Engineering and Information Technology, 2(2010)1, pp. 60-63.

[20] T. Dillon, C. Wu, E. Chang, Cloud Computing: Issues and Challenges, 24th IEEE International Conference on Advanced Information Networking and Applications, IEEE, 2010.

[21] J. Hamilton, Presentation at 2nd Large-Scale Distributed Systems and Middleware (LADIS. Inf. Serv.) Workshop, 15th to 17th September 2008, White Plains, NY.

[22] L. Leong, R. Bala, C. Lowery, D. Smith, Magic Quadrant for Cloud Infrastructure as a Service, Worldwide, Gartner, 15th June 2017. Available: https://www.gartner.com/doc/reprints?id=1-2G45TQU&ct=150519&st=sb

[23] A. Fox, Cloud Computing – What's in It for Me as a Scientist?, Science, 331(2011)6016, pp. 406-407. Referenced on 14th January 2018. Available: http://science.sciencemag.org/content/331/6016/406

[24] M. Hall, G. Zachary, Microsoft Corporation, Encyclopedia Britannica. Referenced on 23rd February 2018. Available: https://www.britannica.com/topic/Microsoft-Corporation

[25] T. Fitzmacken, R. Squillace, K. Crider, Azure Resource Manager vs. classic deployment: Understand deployment models and the state of your resources, Microsoft, 15th November 2017. Available: https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-manager-deployment-model

[26] K. Singh, C. Nottingham, Charwen, Platform-supported migration of IaaS resources from classic to Azure Resource Manager, Microsoft, 10th October 2017. Referenced on 12th February 2018. Available: https://docs.microsoft.com/en-us/azure/virtual-machines/windows/migration-classic-resource-manager-overview

[27] G. Iyer, Cloud Testing: An Overview, Encyclopedia of Cloud Computing, John Wiley & Sons, 2016.

[28] M-Files, the website of M-Files Oy. Referenced on 24th July 2017. Available: http://www.m-files.com/

[29] P. Pietarila, Se oikea löytyi sadan sijoittajan joukosta, Kauppalehti. Available: http://www.kauppalehti.fi/uutiset/se-oikea-loytyi-sadan-sijoittajan-joukosta/xASJuduQ/

[30] Getting Started With Azure Automation – Runbook Management, Azure blog, Microsoft. Referenced on 22nd November 2017. Available: https://azure.microsoft.com/en-us/blog/azure-automation-runbook-management/

[31] M. Balliauw, Introducing TeamCity Azure plugin – Run builds in the cloud, Teamcity blog, Jetbrains, 3rd November 2014. Available: https://blog.jetbrains.com/teamcity/2014/11/introducing-teamcity-azure-plugin-run-builds-in-the-cloud/

[32] D. Tretyakov, TeamCity brings Azure Resource Manager support, Teamcity blog, Jetbrains, 25th April 2016. Available: https://blog.jetbrains.com/teamcity/2016/04/teamcity-azure-resource-manager/

[33] Reuse allocated vms in clone behaviour, Jetbrains Teamcity Azure plugin repository, GitHub. Referenced on 11th February 2018. Available: https://github.com/JetBrains/teamcity-azure-plugin/issues/34

[34] Start/stop with resource manager, Jetbrains Teamcity Azure plugin repository, GitHub. Referenced on 11th February 2018. Available: https://github.com/JetBrains/teamcity-azure-plugin/issues/37

[35] Hamachi, the website of Hamachi. Referenced on 14th April 2018. Available: https://www.vpn.net/

[36] H. Haldane, Microsoft Azure Security, SaaSplaza Microsoft Cloud Solutions, 2015. Referenced on 14th April 2018. Available: http://www.saasplaza.com/blog/microsoft-azure-security

[37] B. Lich, J. Hall, BitLocker, Microsoft, 2017. Referenced on 12th April 2018. Available: https://docs.microsoft.com/en-us/windows/security/information-protection/bitlocker/bitlocker-overview

[38] Virtual Machine Pricing, Microsoft. Referenced on 9th October 2017. Available: https://azure.microsoft.com/en-us/pricing/details/virtual-machines/

[39] Price reductions on L Series and announcing next generation Hyper-threaded virtual machines, Azure blog, Microsoft, 3rd April 2017. Referenced on 9th October 2017. https://azure.microsoft.com/en-us/blog/price-reductions-on-l-series-and-announcing-next-generation-hyper-threaded-virtual-machines/

[40] VPN Gateway pricing, Microsoft. Referenced on 15th April 2018. Available: https://azure.microsoft.com/en-us/pricing/details/vpn-gateway/

[41] IP Addresses pricing, Microsoft. Referenced on 15th April 2018. Available: https://azure.microsoft.com/en-us/pricing/details/ip-addresses/

[42] Teamcity, the website of Jetbrains. Referenced on 30th March 2017. Available: https://www.jetbrains.com/teamcity/

[43] M. Pradeep, Understanding the temporary drive on Windows Azure Virtual Machines, Azure Support Team Blog, Microsoft, 6th December 2013. Referenced on 19th February 2018. Available: https://blogs.msdn.microsoft.com/mast/2013/12/06/understanding-the-temporary-drive-on-windows-azure-virtual-machines/

[44] R. Kumar, I. Foulds, High-performance Premium Storage and managed disks for VMs, Microsoft, 27th June 2017. Referenced on 19th February 2018. Available: https://docs.microsoft.com/en-us/azure/virtual-machines/windows/premium-storage

[45] J. Roth, M. McKittrick, T. Pratt, W. Eastbury, R. Squillace, I. Foulds, J. Molnar, B. Harvey, L. Vargas, Performance best practices for SQL Server in Azure Virtual Machines, Microsoft, 19th April 2018. Referenced on 22nd April 2018. Available: https://docs.microsoft.com/en-us/azure/virtual-machines/windows/sql/virtual-machines-windows-sql-performance

[46] Managed Disks pricing, Microsoft. Referenced on 17th April 2018. Available: https://azure.microsoft.com/en-us/pricing/details/managed-disks/

[47] T. Myers, C. Lin, R. Shahan, T. Pratt, Get started with Azure Blob storage using .NET, Microsoft, 27th March 2017. Referenced on 19th February 2018. Available: https://docs.microsoft.com/en-us/azure/storage/blobs/storage-dotnet-how-to-use-blobs

[48] Azure Service Fabric, Microsoft. Referenced on 22nd April 2018. Available: https://azure.microsoft.com/en-us/services/service-fabric/

[49] M. Marks, Image2Docker: A New Tool for Prototyping Windows VM Conversions, Docker blog, 2016. Available: https://blog.docker.com/2016/09/image2docker-prototyping-windows-vm-conversions/