Lappeenranta University of Technology

School of Business and Management

Degree Program in Computer Science

**Veli-Ensio Heiniluoto**

# Roadmap for .Maintain Framework

Examiners: Professor Ossi Taipale

Supervisors: Professor Ossi Taipale

# TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto
School of Business and Management
Tietotekniikan koulutusohjelma

Veli-Ensio Heiniluoto

**Roadmap for .Maintain Framework**

Diplomityö

2016

48 sivua, 3 kuvaa, 1 taulukko

Työn tarkastajat: Professori Ossi Taipale

Hakusanat: DevOps, jatkuva toimitus

Tämän työn tarkoituksena on laatia suunnitelma .Maintain-sovelluskehyksen toteuttamiseksi. Sovelluskehys tulee tukemaan ohjelmistokehityksen käyttöönoton jälkeisiä toimintoja. Uudet jatkuvan toimituksen kehitysmenetelmät painottavat tätä vaihetta. Tämä vaihe myös tuottaa suurimman osan ohjelmiston elinkaaren kuluista. Tuottamalla työkaluja, jotka tukevat näitä toimintoja voidaan saavuttaa suuria kustannussäästöjä. Sovelluskehys tulee auttamaan uusien ominaisuuksien kehittämisessä, uusien vaatimusten löytämisessä sekä virheiden etsimisessä ja korjaamisessa. Työssä esitetty sovelluskehys koostuu kahdesta osasta: analysaattorista, joka tuottaa informaatiota sovelluskehittäjille yhdistelemällä useita datalähteitä sekä ohjelmointikirjastosta, joka tukee datan keruuta sekä muita toimintoja. Tämä työ toimii lähtöpisteenä Lappeenrannan teknillisen yliopiston projektille, jossa tarkoituksena on laatia kyseinen sovelluskehys.

## ABSTRACT

Lappeenranta University of Technology
School of Business and Management
Degree Program in Computer Science

Veli-Ensio Heiniluoto

**Roadmap for .Maintain Framework**

Master's Thesis

48 pages, 3 figures, 1 table

Examiners: Professor Ossi Taipale

Keywords: DevOps, Continuous Development

The purpose of this thesis was to create a roadmap for a .Maintain framework. The framework would support in tasks performed after initial deployment of an application. This phase, which produces majority of total development costs, is emphasized by emerging continuous development methods. By creating tools that support post initial deployment phase significant cost reductions can be achieved. The resulting framework is based on literature review done on relevant subjects. The framework supports in developing new features, revealing additional requirements, fixing and identification of defects. It is composed of two key parts: analyzer which provides valuable information for developers by combining various data sources and a programming library which supports in various operations including data gathering. The roadmap acts as a starting point for a project started at Lappeenranta University of technology that aims to create a .Maintain framework that can be attached directly to an application at the implementation phase.

## ACKNOWLEDGEMENTS

This thesis was done for the school of Business and Management at Lappeenranta University of Technology.

First, I want to thank my supervisor Professor Ossi Taipale for introducing me to this interesting subject, I learned a lot during the project. Also thanks for motivating me when the work was stagnated and being patient.

I am grateful for all my friends and family for your support. Especially Isto for advice and help during the project.

Big up for those who are yet to graduate! Antti and Simo, now you guys at least have your name in a master’s thesis.

And finally, I want to thank my university, LUT, it was alright after all!

At Lappeenranta on 5.12.2016

Veli-Ensio Heiniluoto

# TABLE OF CONTENTS

## LIST OF SYMBOLS AND ABBREVIATIONS

| | |
|---|---|
| ARM | Application Response Measurement |
| CI | Continuous Integration |
| CD | Continuous Deployment |
| DevOps | Development and Operations |
| FDD SDK | Feature Driven Standard Development Kit |
| FG | Filling-the-Gap (a tool for DevOps) |
| IaaS | Infrastructure-as-a-Service |
| IDE | Integrated Developing Environment |
| OEC | Overall Evaluation Criteria |
| PaaS | Platform-as-a-Service |
| QA | Quality Assurance |
| TDD | Test Driven Development |
| TOC | Total Cost of Ownership |
| TOSCA | Topology and Orchestration Specification for Cloud Application |

# 1 INTRODUCTION

The application development methods are in a constant evolution (Olsson, et al., 2014). Methods that are providing shorter release cycle and faster reaction to customer feedback are gaining traction. There can be seen a trend where companies are continuously deploying program code on a frequent basis, if not daily. Another noteworthy phenomenon is that application testing is increasingly done by utilizing customers as testers. The focus in development is shifting to actions performed after initial deployment. This notion is further supported by analyzing costs related to the development of an application. According to report by Gartner, 92% of the total cost of the application lifecycle is produced after the initial deployment (Kyte, 2012). For these reasons there appears to be an increasing need for tools that support this continuous development of applications. By focusing to the parts of the application lifecycle where most of the costs are produced, significant cost reductions may be achieved.

In this thesis a roadmap for the implementation of a .Maintain framework is proposed. The framework will support in actions performed after the initial release of an application. These tasks include, but are not limited to, development of new features and identification of defects. This framework will be attached directly to an application that is at development. The roadmap is based on the studies done on related fields, relevant standards and technological solutions. Subjects that inspire the roadmap include DevOps, Continuous deployment and lean philosophy. The roadmap is constructed according to design science research method. This paradigm emphasizes the use of existing knowledge and literature in the construction of new artifacts.

Researchers at Lappeenranta University of Technology have started a project called TUTL that aims to build a .Maintain framework. This thesis acts as one the first steps for the project.

## 1.1 Goals and delimitations

The goal for the .Maintain framework is to reduce costs associated to application development. Another important task is to shorten development cycles for the applications and new features. This is achieved by providing tools for development that happens during the maintain phase of the application lifecycle. The question this thesis is attempting to answer is: what would such a framework consist of and how should it be build. During the research central aspects of the development that takes place after initial deployment are expected to be identified. The Roadmap should contain recommendations for the framework, including functionalities it should consist of and technologies that should be used. Possible architectural solutions should also be considered. In accordance to research science paradigm the resulting roadmap should be evaluated in the thesis.

The roadmap should act as a guideline for the further research, providing inspiration for the construction of the framework. Focus is strictly on the actions that are performed during maintain phase of the development.  In depth design decisions and requirement analyses are out of scope of this thesis. The implementation is left for the following iterations of the project.

## 1.2 Structure of the thesis

The thesis starts by reviewing literature from relevant fields. In the review, papers discussing DevOps, continuous integration, continuous deployment, agile methods and lean philosophy are included. Purpose of the review is to gain knowledge that can be utilized in construction of the roadmap for the framework.

The third chapter is about the theory that explains the evolution of the software development methods phenomenon. This shift is a force that creates the need for the .Maintain framework.

The results chapter introduces the recommendations for the framework. The chapter discusses about the four main functionalities that are in center of framework’s operation. Also other design points and architectural options are explored.

The fifth chapter evaluates how the roadmap and the proposed framework succeeds in satisfying the requirements placed on them. Framework’s ability to support in tasks that take place after initial deployment of the application is evaluated. Also under discussion is how the roadmap realizes the philosophies that were used to inspire it. Finally challenges and limitations related to functionalities and architecture are discussed.

# 2 LITERATURE REVIEW

There is no research done on the subject of test and deploy frameworks, but the basis for the framework can be created by closely studying related fields. There exists a number of technologies for individual tasks such as automated deployment (Wettinger, et al., 2014). Some inspiration can be drawn by inspecting the solutions created by companies working with similar problems. In this chapter research and case studies done on the related fields are reviewed.

DevOps (Development and Operations) is a software development philosophy which emphasizes cooperation between software developers and administrators in order to make development more efficient and to reduce defects in the final product. DevOps philosophy calls for automation of its processes, including data collection, distilling knowledge from the data and communications (Cois, et al., 2014).

In continuous integration (CI) development model developers frequently integrate their code against central repository. CI model removes the lengthy task of integration from the software development process. All of the developers are working on the same version of the software (Claps, et al., 2015). Continuous integration requires high degree of tools in order to create an automated building process. Goal is to produce self-testing code that is automatically integrated (Fowler, 2006).

Continuous deployment (CD) can be defined as an ability to deploy software at will. In practice this means that companies are automatically releasing parts of the software into some production environment as soon as they are finished.  As organizations are starting to release code more often, the number of defects reported by customers seems to be decreasing. To achieve this CD utilizes a set of tools ranging from build scripts to automated testing (Fitzgerald, et al., 2014).

Lean philosophy and agile development methods have affected aforementioned practices (Fitzgerald, et al., 2014). Lean philosophy tries to make development work more efficient by breaking process into individual steps and identifying the ones that bring value, everything else is disregarded. Then, the goal is to improve the remaining steps. The agile method Kanban, takes this even further by removing the fixed steps and focusing on continuous work flow (Corona, et al., 2013).

## 2.1 DevOps

In an article released at the communications of the ACM, practices inspired by DevOps are inspected (Roche, et al., 2013). There is no standard definition for DevOps, often it is seen as a combination of developing and operational skills. Other common definition is that DevOps is a new criteria for software development, combining areas such as development, testing, release, support and data gathering. One of the central points in the DevOps philosophy is the utilization of operational data. Crash reports and user data collection are examples of this practice. In the essence is capturing client insight (including what customer sees, when and how often) and turning it into valuable information for software developing. This insight can be used to guide operations such as bug prioritization, test planning and release planning.

CloudWave is an *"execution analytics cloud infrastructure"* which utilizes DevOps principles in order to provide a developing environment which enables deploying continuously improving applications and optimization of the operation environment. It is inspected in depth in a research paper (Bruneo, et al., 2014). Operation of CloudWave is based on three pillars: execution analytics, coordinated adaptation and feedback-driven development.

The execution analytics framework integrates real time usage information from various sources. Resource and sensor information is gathered from data centers. Run time data from applications is also collected, including operational information and user interaction patterns. When combined this data forms a complete view of the operation of the application. It can be used as a basis for decisions regarding future development. To support data analyzing CloudWave provides two different methods: Programmable monitoring and online data abstraction which utilizes various technologies to reduce the amount of data.  Second important function of CloudWave is the coordinated adaptation, which enables cloud environment to dynamically adopt to changes. Adaption covers all the layers of the environment and it is based on a data gathered by execution analytics framework.

Third innovation by CloudWave is called feedback driven development which lets developers to exploit application run time data to steer the development of the software inside agile feedback loops. Tools that are enabling this include: Feature Driven Development Kit (FDD SDK), feedback reporting services and methods for testing effects of software transformations on application's quality and performance.

A scenario demonstrating the effectiveness of CloudWave was provided by the developers, in which developers were working on a cloud based mobile application at a health care organization. At one point they noticed that the cloud was struggling to allocate services. By using run time data received from the feedback mechanisms the root of the problem was identified to be the encryption algorithm, which was then modified. This is one example how these techniques together result in shorter development cycles by letting developers continuously identify modifications that will deliver result on investment.

In a research paper a generalized model for automated DevOps is proposed (Cois, et al., 2014). Purpose of the model is to help disseminate information more efficiently, while reducing the number of defects in the code. Model also frees human actors by automation of the communication tasks. Operations and technologies used for achieving DevOps were identified to be: source control, issue tracking, code review, build, monitoring and communications systems, and an integration environment.

One of the central components of the generalized model is an automated build system. The system monitors the software repositories and acquires the relevant artifacts. Then it builds and performs tests on the software. Deployment is handled by transferring the code to the integration environment. The build system also notifies the communications systems when deployment is ready.

Researchers from University of Stuttgart describe how different DevOps artifacts can be integrated and transformed into TOSCA standard model to realize automated end to end deployment in the cloud environment (Wettinger, et al., 2014). There exists a number of technologies for an automated deployment of an application, these packages include all the necessary scripts and modules. This paper inspects two of them: Juju charms and Chef cook books. A single technology might not contain all the required functionalities thus creating a need for integration. However these technologies require their own run time environments making integration difficult. Researchers developed a solution utilizing TOSCA (Topology and Orchestration Specification for Cloud Application) standard. TOSCA creates a unified meta-model, that can be used to integrate different technologies into cloud infrastructure.

Chef is a configuration management framework. Its operation is based on bundles of configuration definitions called cookbooks. Cookbooks contain all the necessary commands needed to deploy an application on a single node, inter node relations are not supported. Commands can be, for an example, installing a MySQL database or an Apache server and configuring them. Chef also contains a tool called knife, which is used to manage components connected into a chef server.

In contrast to Chef, Juju is a tooling dependent solution for deploying applications on a multi node environment. Juju uses script files called charms. Charms contain a complete lifecycle of a component. Charms consist of commands such as: starting and stopping a database server. Juju also contains a management node used to control charms.

FG (filling-the-gap) is a tool that implements DevOps by providing methods for collecting and analyzing operational data (Perez, et al., 2015). FG was developed for two purposes. First one is to provide data for improvement of quality of service models. Second one is to provide reports about application's runtime behavior. FG has a framework that is able to measure both application and system level metrics. Other important components include a monitoring history database and a design time component used to update quality of service models.

FG is able to monitor four different parameters. Measuring tools were designed with the purpose of evaluating service level objects in mind. First one is the current user population of the application. The data is based on a total number of requests performed on the system. To obtain the number of requests a monitor for each of the applications main methods is needed. Second parameter is resource consumption (CPU). For measuring this, various methods can be used: CPU throughput, length of queues at the resources and response times. Third parameter to be monitored is think time meaning the time user spends inactive using the application. It can be obtained by indirect measurements, utilizing total and mean number of requests performed on the system. Also, if available, request arrival rate can be used. Fourth parameter, stage duration is related to deployment of an application. These stages describe the state of the resources on which the application is deployed. They can be used for measuring efficiency of the deployment. FG requires a monitoring platform to function, for an example MODACloud is sufficient.

Logs produced by an application can also be used to implement DevOps practices (Shang, et al., 2012). In the study researchers analyzed logs produced by varying applications, findings concluded that logs contain a rich source of information beneficial for both developers and operators. Researchers also proposed ways to utilize logs more efficiently.

Developers can use logs to identify error-prone components, or classes, in the software. To achieve this logs that are updated often are identified. Then metrics, including code complexity and number of pre-release bugs, are included into a statistical model in order to find out classes that are prone to errors. Logs can also be used to evaluate field test coverage for systems. In practice this means creating two different models from logs, field execution model and testing execution model, and comparing them to calculate the test coverage.

Logs can be used to reduce operational complexity, two ways were proposed. Currently there exists no standardized way to document application logs. Study suggests that by attaching development history and bug reports to corresponding lines logs can be utilized more efficiently. Logs are constantly changing during application development, developers need to inspect these changes in order to figure out their impact. By analyzing source code, filters can be constructed to bring out the changes to support development.

## 2.2 Continuous Deployment

Continuous Deployment (CD) can be defined as a practice that takes continuous integration even further by automating following tasks: deploying to testing environment, acceptance testing and deploying to production environment. While continuous integration only automates code change detection, unit testing and integration testing (Pulkkinen, 2013). The seminar paper also introduces continuous deployment strategies and tools common in the industry.

To achieve CD an automated build pipeline must be implemented. This pipeline consists of all the necessary tools needed to deploy a piece of software from commit to production environment. Essentials tools are described in table 1.

**Table 1. Essential tools for an automated deployment pipeline (Pulkkinen 2013)**

| Application type: | Application examples: |
|---|---|
| Version Control System | Git, Mercurial |
| Continuous Integration Server | Jenkins, Hudson |

| | |
|---|---|
| Software Configuration Management | Chef, Juju |
| Automated Test Suites | Junit, JMeter |
| Database Change Management | DbDeploy, Liquibase |
| Build Tool and Dependency Management System | Apache Maven, Rake |

Because the pipeline forces developers to follow good and efficient developing practices, it is concluded in the Continuous delivery book (Humble, et al., 2010) that:

> *"Even if it is not possible to apply the continuous deployment as your software development strategy, you should build your build-pipeline as such as you could switch to continuously deploying every commit to production at any time."*

Author of the paper also describes common strategies used to reduce risks related to deployment. Feature flags are used to dynamically toggle on and off software features if they are causing issues. They can be implemented on the code level. Dark launches can be used to hide software features from actual users after deployment, only testers are granted access to said features. This method should only be used for performance testing, as other tests should have been performed during deployment pipeline. In blue-green method two different environments are utilized: first environment contains the original working version of the software, while second environment has the newly deployed one. If issues occur system can be restored by falling back to the first environment. Canary releasing is testing strategy where only a small portion of actual users is exposed into a new feature. It is related to A/B-testing practices where selected group of users is split into two groups. These groups are exposed to two variations of the same feature.

Author also introduces various solutions to implement a CD environment. Heroku is a platform-as-a-service (PaaS) solution that automatically builds and deploys pieces of software as they are committed to its repository. Continuous integration environment could be used to commit changes to a Heroku repository when the code has passed the automated tests. Heroku provides this service as an add-on, Tddium. Red Hat OpenShift is another PaaS solution that implements continuous deployment, its operation is based on a popular continuous integration server Jenkins. OpenShift functions in a similar way to Heroku. However, unlike Heroku parallelization of tests is not supported. Finally author proposes combining a PaaS continuous deployment solution with an integrated developing environment (IDE) that supports collaborative programming to achieve even faster development process.

In a survey performed by North Carolina State University most common practices related to continuous deployment were identified (Rahman, et al., 2015). Most used practices include: automated deployment, automated testing, code review, dark launching, feature flags, monitoring, repository use and staging. Survey also included most used testing practices: unit, integration, A/B, acceptance, regression and functional testing.

Perceptional testing is an emerging practice that tries to identify non-trivial errors invisible for human eye. These errors are seen as a source for customer dissatisfaction. In perceptual testing snapshots of two different versions of the user interface are compared pixel by pixel. Visual difference of the pages is then calculated.

Case study (Neely, et al., 2013) reviews Rally Software's transition to a continuous delivery model, and also introduces the rationale for the change. Before transitioning to CD, the company was using the Scrum development method, with an 8-week cycle. This cycle was felt as a limiting factor as precious features would have to wait until end of the cycle. If deadline was missed a feature would have to wait another eight weeks. Smaller batch sizes were also expected to reduce the number of the defects and make integration easier. Continuous deployment allowed developers to test new ideas easier. Kanban methodology was used to replace Scrum, as it allows more continuous workflow.

A number of continuous deployment practices were used including: feature toggles, dark deployments and canary deployments. To implement dark deployment programming library AKKA was used. Feature toggles were also utilized to grant access to new features for specific users. Practice of testing code manually had to be changed, however quality assurance personnel were still needed to prioritize tests at pre-development phase. In conclusion Rally software saw a significant increase in code throughput and a decrease in a number of reported defects. Also occurrence of defects was noticed to be more predictable than before.

The social media company Facebook created a framework to support continuous development (Feitelson, et al., 2013). Some companies let developers release code straight for users, but since Facebook operates with highly confidential data, more sophisticated deployment method is used. One of the most important aspect of the Facebook's model is a live testing method, called A/B-testing, which facilitates actual users as testers. In A/B testing developers select a small subset of users, and release the new functionalities for those users. By closely monitoring the actual user's experience, developers can see what works and what does not. Facebook pushes new pieces of code in small increments to reduce the risks related to deployment. They have implemented a tool called Gatekeeper which is used to control user's access into different versions of the code. It is a tool that enables A/B testing. Each new piece of code is automatically regression tested in order to detect any bugs that could emerge at the system. Version control system Git is utilized to keep track of the code. Stability of each code branch is closely maintained.

The deployment process at Facebook is strictly defined, it consists of three steps, in which each piece of code is rigorously inspected. At the first step code is released to internal Facebook servers where final tests are performed. In the second phase code is released for small fraction of real world users. If any problems are found they will be fixed and the cycle repeats itself. After code passes these steps the final deployment phase begins. Facebook uses Bittorrent technology to transfer the final version to various servers around the world. At the time of the code push all engineers responsible for the code must be available online, in case issues occur. A system that utilizes IRC-bots is used to achieve communication between engineers. Facebook closely monitors their systems to detect bugs and issues, a system health monitoring software is used. Data is gathered from both internal sources and external sources, such as Twitter.

According to case study the step after continuous deployment is seeing research and development as an experiment system (Olsson, et al., 2014). At this stage the entire system is able to respond to customer feedback. Key aspect at this stage is using the system to reveal customer requirements by experimenting and testing. Advanced instrumentation for data collection is needed. Organization also needs the capability to effectively use the collected data.

In the study organization’s employees were interviewed about their experiences in moving towards continuous deployment. Employee in one company highlighted a need for data about deployment. Developers would benefit from information achieved from the deployment pipeline including the current quality of features and the number of errors, this would allow teams to increase the quality of the subsequent builds. One major challenge in continuous development was the wide variety of configurations that customers had. This made the deployment of new features a tedious process. Another challenge experienced was the wide variety of deployment related tools, developers would benefit from well-defined processes and tools.

## 2.3 Continuous Integration

Experiences implementing a continuous integration environment are described in (Abdul, et al. 2012). The process begins by defining a build strategy best suited for the application at hand. Then appropriate tools are selected, modern IDEs usually include functionalities to develop build scripts. The actual build is handled by an integration server such as Jenkins or Hudson. Other tools frequently used in the CI systems include: bug tracking, version control system and testing tools.

The CI process starts by gathering all the required pieces of code from repositories. Complexity of this task depends on the amount of repositories. Next step is a build process that can include tasks such as compilation, unit testing and code obfuscation. Packaging is the final step of the process. It includes activities related to making the product distributable to other parties. Depending on the needs such as compatibility requirements the package can an executable or an archive file.

Central concepts of CI are reviewed in (Fowler, 2006). Continuous integration requires a high degree of automated tests embedded into code, sometimes called self-testing code. There exists a discipline studying these practices called Test Driven Development (TDD). To produce self-testing code a suite of automated tests is needed, and only when tests are passed the build can begin. A testing framework called xUnit can be used for this purpose, there are also other tools including FIT, Selenium and FITnesse.

Agile testing techniques can be used together with continuous integration model (Stolberg, 2009). One of the most important tasks is to define the acceptance tests and automate them to highest possible degree, to guarantee that customer requirements are met. This also reduces the amount of regression testing required. xUnit framework can be used for acceptance testing.

## 2.4 Agile Methods and Lean Philosophy

Lean philosophy can be applied to a design of software to achieve high testability (Alwardt, et al., 2009). Testability is significantly affected by two factors: cohesion and coupling. Cohesion measures how well functionalities inside model are related to each other; high cohesion is desirable. Coupling measures how inter-connected modules are, low coupling makes testing easier. However with complex systems coupling cannot be completely avoided, modules with high coupling are not unit tested. With complex systems regression testing should be preferred. Unit tests and regression testing should be kept separated.

Dependency injection is a technique used to enable unit testing in highly coupled systems. In this practice mock objects are used to replace external dependencies. After unit tests have passed the system can be integration tested.

Setup and teardown methods can be utilized to enable automated testing. Setup method is used to prepare system for testing by returning it to a known state. Tear down method returns system back to its original state after tests are run. Usually these methods are needed when databases or singleton objects are present.

Lean 123 initiative introduces a three point checklist for execution of tasks. It can be used when designing software testability. The checklist is: Establish clear priorities, eliminate bad multitasking and limit the release of work in process. Automated regression tests can be used to enable lean approach in software engineering (Writing software that tests software). Lean also emphasizes reducing waste, in software testing this can be achieved by planning a minimum number of tests, just enough to meet the requirements.

The agile method SCRUM, the type-c version, can be modified to be used in the continuous delivery model (Agarwal, et al., 2011). In type-C SCRUM sprints of varying lengths are overlapping. Weekly sprints provide bug-fixes, monthly sprints are for features and quarterly sprints are for major enhancements. Sprints are performed by multiple teams. In the continuous SCRUM model there are three sprints: planning, development and QA. A single team performs all the sprints simultaneously. In this model deployment into production environment will happen weekly.

The key to achieve weekly releases is a build and deployment infrastructure, as proposed by the authors. The infrastructure consists of following parts: automated build scripts upon commit, developing server for peer testing, controls for moving work item between environments and controls for releasing code in different phases. Authors promote the use of automated testing, including automated user interface testing and user input recording.

Software company IMVU applied lean principles to their development process. Number of technical artifacts were implemented to achieve this (Widman, et al., 2010). The project was highly successful, authors conclude that the key element in applying lean principles to software development is a comprehensive testing environment.

First lean principle IMVU utilized was “Specify Value in the Eyes of the Customer”. By releasing sub-par product as soon as possible they were able to steer the developing work according to user feedback, thus reducing waste in work hours. Second principle: “Identify Value Stream and Eliminate Waste” was realized by implementing a continuous deployment pipeline. It allowed engineers to identify problems related integration sooner than using traditional methods. “Make Value Flow at the Pull of the Customer” is the next principle that benefited IMVU. All the new features were at first tested with a small number of actual users. Using this method it was possible to identify features that would not be successful and then cancel development of them. In accordance of “Involve and Empower Employees” - principle an individual copy of developing environment was created for each employee. In the sandbox developers were able to test their code using a set of automated unit, acceptance, functional, and performance tests.

## 2.5 Micro services

Micro services is a cloud architecture where application is divided to small independent parts that each provide a service (Savchenko, et al., 2015). These services communicate with each other using messages. They can be duplicated and moved to any other computational resource. Services can be complex software systems containing local storages or web servers for an example. Automated deployment is used in the development of micro services. Authors describe a set of tests to be used when adding new services to a micro service architecture. These tests are related to inter service communications. First one is functional integration validation where inter-communication of services is tested. Second testing method is load integration validation which checks service’s correctness during automatic deployment. It includes the task of finding the maximum communications load a service can handle. Last method is integration security validation which checks the security and robustness of inter service communication. In addition micro service communication interface should be validated.

## 2.6 Summary

Numerous solutions and tools for testing and deployment were identified from the literature. There seems to be no significant gap. However there is no solution that would combine the tools, providing an easy to use solution for developers. Selection of testing methods is also dependent on the application's architecture, for an example micro services model introduces a new set of tests. Following features for testing and development were identified:

- deployment scripts (Wettinger, et al., 2014) (Pulkkinen, 2013)
- run time and application data collection (Perez, et al. 2015) (Roche, 2013) (Bruneo, et al., 2014)
- log analytics (Shang, et al., 2012)
- feature flags, dark launches, blue-green method, canary releasing and A/B-testing (Pulkkinen, 2013)
- unit, integration, functional, acceptance, perceptual and regression testing (Rahman, et al., 2015)
- dependency injection, setup and teardown methods (Alwardt, et al., 2009)
- automated user interface testing (Agarwal, et al., 2011)
- functional integration, load integration and integration security validation tests (Savchenko, et al., 2015).

In (Pulkkinen, 2013) a continuous development environment is proposed. It contains all the tools needed to push a piece of software from commit to deployment. In the core of its operation is a PaaS solution for CD, such as RedHat OpenShift, combined to an automated testing suite that contains tests identified from the literature. However, tools for monitoring application's run time activity are not included.

# 3 THEORY

There can be seen a constant evolution of software development practices. Methods become more agile in response to requirements rising from the market situation. The focus in application development is shifting more and more to actions performed after initial release. This can also be seen by analyzing the cost of ownership of an application. In this chapter these phenomena are explored. Last section of this chapter introduces design science, the research method used at the construction of the roadmap for the framework.

## 3.1 Costs of Software development

According to a report from research company Gartner application's total cost of ownership (TOC) consists of four components (Kyte, 2012):

- cost of initial project
- cost to operate
- cost to support and maintain
- cost to enhance and extent

Cost of the initial project is 8% of the total cost, this includes tasks such as requirement analysis, design and implementation. Last 92% of costs are produced after the initial release, actions taken at this phase include: introducing new features and fixing defects. TCO is determined to be an outcome of design decisions and life cycle management decisions. Costs are distributed unevenly during the lifecycle and will tend to increase exponentially over time for large projects. In order to reduce TCO Gartner gives two recommendations: investments to maintainability should be made and applications should be designed having the whole lifecycle in mind. This leads to a change of mindset where instead of thinking maintenance as process of making minor enhancements, it should be seen as series of corrective, preventive, adaptive and perfective actions (Kyte, 2012).

## 3.2 Shift in Developing Model

The software developing models are constantly changing. Evolution of software development practices can be seen as a path that gradually leads from traditional development methods to agile methods and then, finally, to continuous deployment and beyond. Figure 1 visualizes the evolution of software development (Olsson, et al., 2014). The reasons for the change in developing model can be traced back to the market situation. Markets along with customer requirements are unpredictable and fast-changing, they are affected by complex factors. Increasing competition also demands faster release cycles. For these reasons development methods with shorter iterations seem appealing. They offer more flexibility and faster reactions compared to traditional methods (Dzamashvili, et al., 2010).

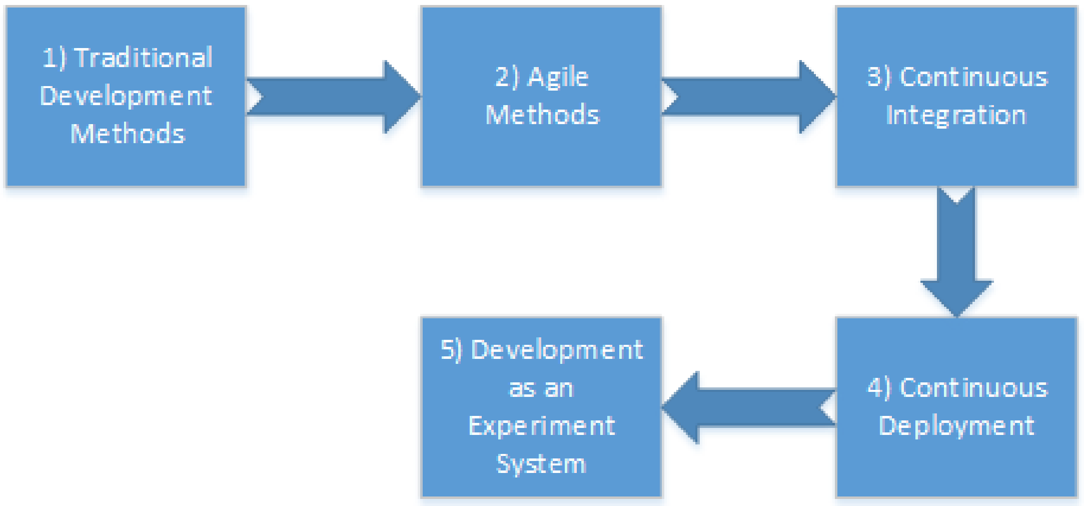
1) Traditional Development Methods
2) Agile Methods
3) Continuous Integration
4) Continuous Deployment
5) Development as an Experiment System

**Figure 1.** Evolution of software development methods

Typically companies have developed software using incremental methods with slow release cycles (1). The methods move from step to step, from analyzing to implementation then to testing and finally ending with the deployment of an application. Mechanisms for reacting to customer feedback are not well integrated to the process (Robillard, et al., 2003). A next step for organization is to start experimenting with agile methods (2) which provide shorter development cycles. But time from received customer feedback to change made to a feature is still relatively long. Method is not considered agile if the release cycle is longer than six months. Typically length of an iteration ranges between two and six weeks. Agile methods, such as extreme programming, emphasize improving design of the application constantly and performing testing as often as possible. Typically at the end of each iteration customer is given a chance to have an effect to requirements. (Highsmith, et al., 2001).

Next step forward from agile methods is moving to continuous integration (3) and deployment (4) (Olsson, et al., 2014). Goal of these methods is that developers can automatically integrate and deploy code frequently if not daily. This practice acts as an enabler for shorter customer feedback loops. At this point research, development, product management and customers are part of same agile development cycle. The final stage of the evolution is seeing research and development as an experimentation system (5), where customer feedback is received instantly, and development can be steered according to it. This leads to a mindset where:

*"Deployment of software is seen more as a starting point for further 'tuning' of functionality rather than delivery of the final product."*

These last steps require not only sophisticated tools to handle automated deployments, but a support and a full involvement from the organizational units and stakeholders including customers. Processes need to be fine-tuned and focus needs to be shifted from components to features.

## 3.3 Design Science

Design science is a research methodology used in information systems research that: “*Creates and evaluates IT artifacts intended to solve identified organizational problems* “. Such structured artifacts can vary from software and mathematical formulas to informal descriptions in natural language (Von Alan, et al., 2004). Another important quality to these artifacts is the relevancy to the previously unsolved problem. According to design science, development of these artifacts should be a search process that utilizes existing knowledge (Peffers, et al., 2007). After the artifact is constructed its quality and utility must be rigorously investigated. Finally the results are communicated to appropriate audiences.

There exists a guideline for conducting a design science project (Peffers, et al., 2007), it consists of six steps, or activities. The process begins by carefully defining the problem, here dividing the problem to smaller sub-problems might prove to be useful. Then the value of the solution must be justified. This step builds on top of the existing knowledge on the problem. The next step is to take the problem definition and to start drawing objectives from it. Objectives can be for an example: In which ways the new solution would be better than the existing ones or which previously unsolved problems would a new solution solve. Knowledge about existing solutions can be used as a basis for this step.

After the solution has been defined the next activity is the implementation of the artifact. Tasks performed during this activity depend greatly on the problem and the artifact. In general they include: defining artifact’s functionality, determining its architecture and then actually implementing it. This step of moving from problem definition to solution requires a strong grasp of the underlying theory. Next activity is to take the artifact and demonstrate its effectiveness by solving one or more instances of the problem. Methods of demonstration include, but are not limited, to experimentation, case study and simulation. This activity requires strong knowledge about the artifact.

After artifact has been demonstrated to solve the problem at hand it is evaluated. Evaluation is based on observed results achieved during demonstration. These results are then compared to objectives drawn during problem definition phase. Evaluation can be performed in numerous ways such as customer surveys, simulations or actual metrics about the artifact's operation. According to evaluation researchers may decide to iterate back to design step and make required modifications to the artifact. If the artifact is satisfactory last step is to communicate the results to appropriate audiences. Relevant information that should be communicated includes: the problem and its importance, the artifact, its utility and novelty, the rigor of its design, and its effectiveness.

The design science research process does not have to start from the first activity. The nature of the problem defines the appropriate starting point. For an example an object centered project, triggered by a needs of industry, may begin from the second step.

# 4 RESULTS

The framework supports in tasks related to data gathering, analytics and testing. Its operation can be divided into four parts that each provide an essential functionality:

- probe
- analyzer
- simulator (dependency injection)
- methods to restrict users access to the parts of the application

Probes are implemented as a programming library, they are attached directly to the application code and are used to gather operational data. This data is fed to an analyzer. The analyzer combines data from various sources and converts it to information that can be used to make intelligent decisions considering the development. The framework also provides other supporting functionalities.

Simulator is here defined as an artifact that is used to replace parts of the application to make unit testing possible. The technique also known as dependency injection is used when the application has a complex structure with a number of inter-connected external resources, such as databases.

One of the important design principles in the framework should be utilizing customer information in testing and development in order for development to move beyond continuous deployment, as described in the case study (Olsson, et al., 2014). New features should be tested with customers, as early as possible, to provide value as most business decisions seem to fail in having any effect on performance (Kohavi, et al., 2009). In addition to gathering user information with probes, methods used to control user's access to parts of the application should be included to support testing and experimentation with new features.

The deployment pipeline should be kept separated from the framework as tools for its implementation are readily available. However the framework should be designed in a way that it can work in cooperation with the pipeline.

## 4.1 Automatic Deployment Pipeline

Continuous deployment is a practice where the system tries to automatically build a piece of code as soon as it has been uploaded into a central repository (Pulkkinen, 2013). If all the tests are passed, the artifact is automatically deployed into the production environment. If any of the tests fail developer receives instant a feedback. Tests that are usually part of the continuous deployment process include unit, integration and acceptance tests. Automatic deployment pipeline is an actual implementation of continuous deployment, consisting of tools and well defined practices. The process is shown in figure 2.

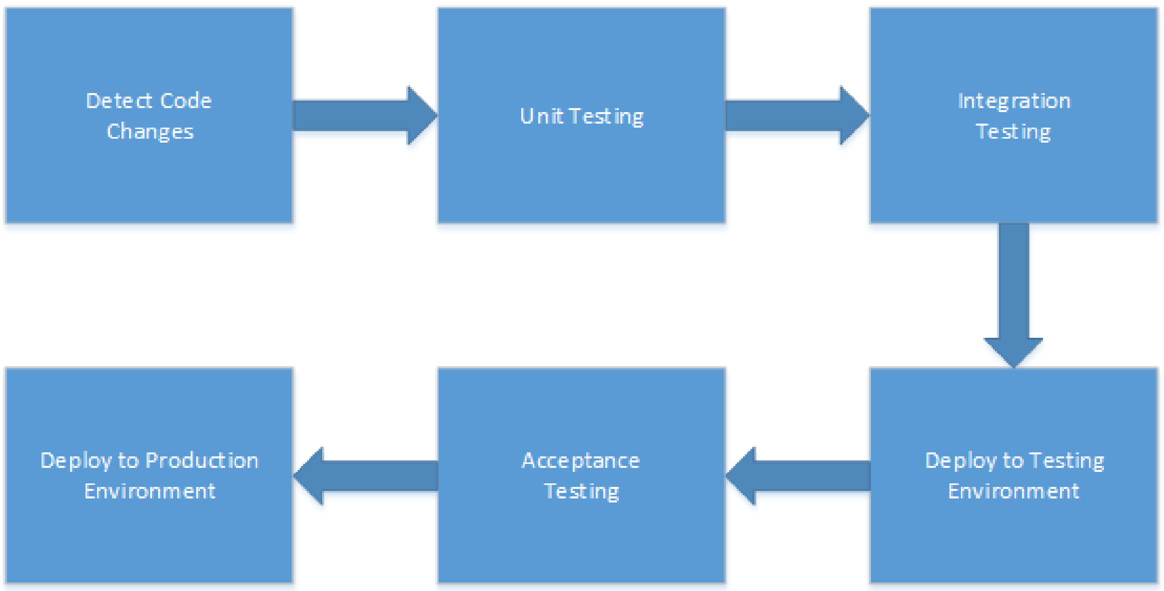

**Figure 2. A Continuous Deployment Pipeline**

Continuous deployment pipeline should be implemented in every production environment because it will help fixing the defects by offering repeatability and traceability, it will also lower risks related to development by forcing developers to release code in smaller increments. Continuous deployment utilizes customers as part of quality assurance and development (Pulkkinen, 2013).

Tools for implementing continuous deployment are introduced in chapter 2, these tools enable a complete implementation of continuous deployment. The pipeline should not be part of the framework. This separation provides the framework flexibility by reducing external dependencies. This also enables more lean design. The pipeline can utilize data from the framework and use it to guide the deployment process. The framework can also benefit from testing and deployment related data provided by the pipeline.

## 4.2 Analyzer and Testing Strategies

Analyzer is a piece of software and a part of the framework that is responsible for transforming data gathered from various sources to valuable information. Inspiration for the analyzer was taken by studying cloud platforms (Caron, et al., 2011) and adaptive and analytic solutions for the clouds (Bruneo, et al., 2014) (Perez, et al., 2015). Analyzer should collect data from:

- probes that are attached to program code
- logs produced by the application
- underlying infrastructure:
    - hardware (memory usage, network usage, CPU usage)
    - virtual machine and cloud environments
- external resources such as databases
- external internet services
- internet of things around the application.

Only by combining data from various resources can a complete view of the application’s operation be formed. Depending on the developed application the amount of the data can prove to be overwhelming, analyzer needs to combat this problem by providing relevant tools. Tools are required for compressing, abstracting and filtering the data. Another method for reducing the amount of data is making monitoring programmable, which enables developers to choose appropriate measures depending on the current application and testing policies.

With faster release cycles due to continuous deployment some parts of testing the components is allocated to customers. One company that relies to this strategy is Facebook (Feitelson, et al., 2013.). A/B testing is one example of customer powered testing. In A/B testing customers are randomly split to two even groups (Kohavi, et al., 2009). One of these groups is exposed to an original version of the program or a control variant. Other group is given a modified version, or a treatment variant. Observations are collected and an overall evaluation criteria (OEC), or a metric, is defined. If experiment is designed correctly any changes to the OEC must be because of modifications done to application that is tested. One of the common OECs is a click through ratio, meaning a number of users that have used the selected functionality. Another way to define OEC is to measure changes in return of investment, for an example when testing design of advertisements. A sufficient data collection both server and client side is essential for A/B testing.

At Microsoft the support team wanted to determine whether making the help pages more personalized would be beneficial. In control variant user were given answers to most common problems from all the different segments. In treatment variant the answers were customized according to customer’s browser and operating system version. The click through ratio for links in both variants was calculated, proving that simple personalization in variant group provided more clicks and value to customers. In a same way A/B testing can be used to test new prototypes. The metrics provide accurate real world information whether customers behaved as expected and whether the added functionality provided any added value.  Analyzer is the part of the framework that acts as an enabler for testing and experimenting performed with customers, thus it should be designed to support novel testing strategies that are not included in the deployment pipeline.

In this chapter design points for the analyzer part of the framework were explored. Analyzer is responsible for:

- gathering and combining data
- enabling programmable data gathering
- compressing, abstracting and filtering the data
- providing methods for novel testing strategies

## 4.3 Data Gathering and probes

Data gathering is necessary component for many testing strategies such as A/B testing, it also simplifies bug detection and guides direction of the application development. Data gathering is the key to realizing DevOps and enabling experimenting with customers. A number of data sources can be utilized. Most common sources identified from the literature include data collected from application and virtual machine levels. Also runtime data from underlying hardware can be exploited. In this chapter various probes and data sources regarding application's behavior are introduced. Inspiration for the probes was drawn by studying self-adapting software and related measurement standards.

Probes should be placed in the application to monitor its state. Probes are implemented as a programming library, they can be attached directly to application code during development. There exists a number of standards defining a set of sensors to gather data from the software entities (Salehie, et al., 2009). Application Response Measurement (ARM) is a monitoring standard:

*"Which enables developers to create a comprehensive end-to-end management system with the capability of measuring the application's availability, performance, usage, and end-to-end response time."*

Other relevant standards and techniques include:

- CBE (Common Base Events)
- WBEM (Web-Based Enterprise management)
- JVMTI (Java Virtual Machine Tool Interface)
- JMX (Java Management eXtensions)
- CIM (Common Information Model)

Some of these are designed for specific programming languages, namely ARM which is bound for JAVA and C languages. In addition sensors can be used with proprietary software where recompilation is not possible WPI's AIDE and OBJS' ProbeMeister are examples of such solutions. Network monitoring is also useful in gaining information about state of the application (Parekh, et al., 2006).

As described in previous sections probes can be used to gain other useful attributes such as user population, think time and response time (Perez, et al., 2015). To achieve these a probe must be attached to each class's main method, obtaining data about methods that are called.

Click through ratio refers a practice of counting number of clicks on a link compared to total number of users who see the page. It's an essential metric in A/B testing (Kohavi, et al., 2009). The click data can be collected at server side, or at client side depending on the environment. In addition to collecting click through information, the data about user's interaction with the page should be gathered, since it can be utilized in various ways to guide design decisions, as seen in the previous chapter's example.

Logs generated by the application can be utilized in testing and development. Researchers proposed a way in which logs can be used to predict error prone classes that could experience bugs in future (Shang, et al., 2012). Logs offer a promising and commonly used data source for the analyzer (Salehie, et al., 2009). One commercial example of log utilization is Microsoft’s Azure cloud platform (Caron, et al., 2011).

Probes described in this chapter can be attached directly into code, although there exists methods to measure already compiled application’s state. Probes should measure:

- user population
- think time
- usage
- availability
- end to end response time
- method calls
- click through ratio and user interaction

Together probes and other data sources including run time data from underlying infrastructure and external internet services form a complete picture of system’s operation. Analyzer is used to combine this data.

## 4.4 Simulator

Dependency injection is a design pattern in which object’s dependencies are substituted with mock objects (Alwardt, et al., 2009). These mock objects are, in practice, fake versions of real objects. They simulate the original object’s behavior by sending hard coded messages or more intelligent responses based on a way they were called. For an example the object could be a database or a web service. These mock objects are switched in during unit testing. However integration testing needs to be performed using real objects in order to test the communication between objects. Dependency injection should be used because it simplifies and speeds up the testing process especially with complex systems.

Certain tasks that a simulator needs to perform are highly prescribed by the programming environment (Ekstrand, et al., 2016). These tasks include identifying dependencies of each component and wiring components together ensuring that all the dependencies are satisfied. There exists a number of tools for each programming language to implement dependency injection.

Dependency injection or a simulator supports unit testing which is a part of deployment pipeline and not a part of proposed framework. There exists a number of tools to implement simulating parts of the software including Picocontainer for Java (Hammant, 2011) and Ninject for .Net environment (Kohari, 2012).

## 4.5 Methods for Controlling User Access

As testing and prototyping new functionalities is performed more on users, sophisticated methods to control user's access to parts of program are needed (Pulkkinen, 2013). They allow only a selected group to be used as testers. Certain testing strategies also require capability to split users to groups. As a new prototype is deployed into production environment there needs to be a method to turn it off in case of unexpected problems.

Feature flags enable developers to turn features on and off, in case any problems occur. They are the requirement for the A/B-testing family. They can simply be implemented in code level. Dark Launches are a technique that enables features being tested in production environment, without customer interaction. Testers interacting with the feature can be automated tests or humans. This way the systems performance can be tested and data about the operation gained.

## 4.6 Architecture

The framework is composed of two key parts: the analyzer and the programming library. Programming library consists of implementations for probes, simulator and user control mechanisms. In figure 3 an architecture for the framework is proposed.

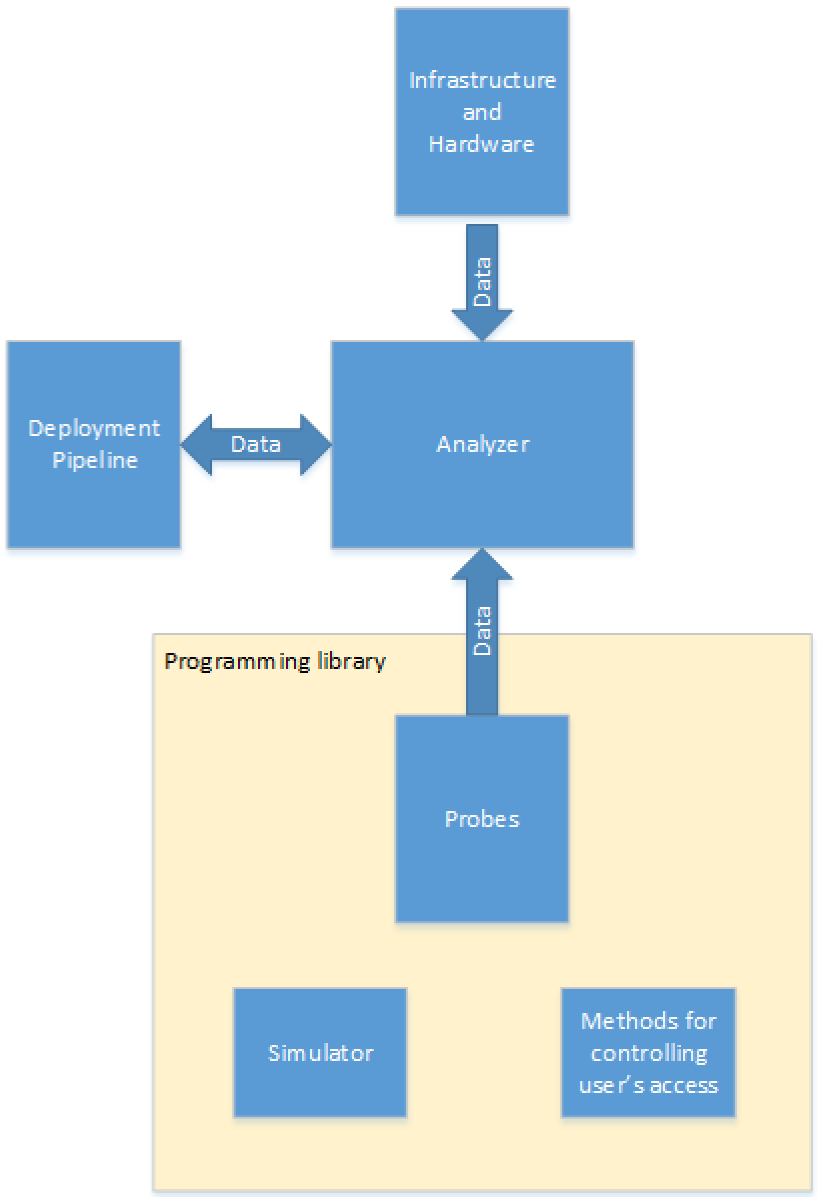

**Figure 3. Architecture for the framework**

Analyzer is a piece of software that gathers data from deployment pipeline, underlying infrastructure and probes. Analyzer can also exploit data from other external resources that are left out of the picture for simplicity, such as databases and application logs. There is a two way relationship between the pipeline and the analyzer, as the deployment process may benefit from operational data. An example of this relationship is using test coverage data derived from application logs to target unit testing in the deployment pipeline. The pipeline can supply analyzer with data about quality of features and number of errors, for an example, as suggested in a case study (Olsson, et al., 2014). Simulator and methods to control user access are functionalities that support development, they have no direct connection to analyzer.

# 5 DISCUSSION

The goal for the framework is to support in tasks related to the maintain phase of software development. Central characteristics of this phase were identified from the literature. In the center of action is adding new functionalities, fixing the defects on existing ones and revealing additional requirements. The framework attempts to form a complete view of the application's operation thus offering a way to identify defects as soon as they appear. Adding new functionalities and revealing additional customer needs is also supported by data gathering. This enables developers to identify new requirements and explore which features bring value for customers and which do not. The framework offers tools that support experimenting with new features, namely the methods to restrict user access to the parts of the application. Utilization of data is also a key aspect in turning research and development to an experiment system.

Reduction of total ownership cost (TOC) of the application was one of the key motivating factors for the framework. According to report by Gartner (Kyte, 2012) design decisions have a direct effect on TOC. Using customers for testing new features and receiving real world usage data supports in making more intelligent decisions that bring more value. As 75% of business decisions fail to provide any value (Kohavi, et al., 2009), there exists a potential for significant reduction of costs.

Other important goal for the proposed framework is to shorten development cycles of the application and new features. This is where probes and other data sources utilized by the analyzer provide value. Developers of FG (Perez et al. 2015), demonstrated the effect of DevOps strategies in a motivating scenario. These methods are expected to provide significant speed ups to the development.

One of the inspirational sources for the framework was DevOps. Which is utilizing operational data in development, and vice versa. The framework provides operational data by the use of probes. The DevOps principle is further realized by the analyzer which combines application's runtime data with the data from the underlying infrastructure.

Analyzer also attempts to turn data into a valuable information for developers by offering various tools related to filtering, abstraction and visualization of data.

Other important aspect in the framework is the support for continuous development of the software. While the framework does not implement continuous deployment pipeline it supports deployment by offering data, this data can be used to guide testing for an example. By keeping the deployment separate the framework maintains its flexibility and modularity, this is further supported by the lean philosophy. There also exists a range of tools for a full implementation of continuous deployment pipeline.

The framework can be seen to offer support to all three central tasks of post initial deployment development that were identified. Additionally it also realizes the two central philosophies that were used as an inspiration. In this chapter roadmap for the framework is further evaluated by looking at its functions and architecture. Last section justifies the roadmap and the framework by explaining the key differences to existing solutions.

## 5.1 Functionalities and Architecture

Probes collect data about application’s operation. They are implemented as a programming library. This approach allows a wide range of data to be collected. However this approach is not all inclusive, data from infrastructure and hardware levels needs to be also collected in order to get a complete picture. For an example some data such as click through ratio can be calculated in a multiple ways: in addition to using a programming library solution, data can be achieved from a server or infrastructure level (Kohavi, et al., 2009). Further investigation is needed to determine which probes benefit from programming library implementation. During research the list of probes presented is expected to change.

Certain tasks are heavily dependent on selected programming language. For an example monitoring standard ARM is bound for C and Java languages. The programming language dependency must be considered at an early design phase. There are existing solutions for multiple tasks included at the programming library. One example of this is technique related to controlling user access, dark deployments. Which can be achieved by the use of programming library called AKKA. It must be evaluated whether new implementation would provide any value.

The simulator is used for replacing parts of the program in order to simplify unit testing. There exists a number of solutions for its implementation. Dependency injection techniques are also heavily dependent on the programming language, there may exist programming language independent simulation techniques, but they were not found during literary review. If included to the framework it would introduce new requirements and dependencies and complicate design. Unit testing is part of continuous deployment pipeline, and then outside of the scope of proposed framework. Further research is needed to determine whether application development would be benefited by the inclusion of dependency injection techniques to the programming library.

## 5.2 Comparison to existing solutions

The proposed framework differs from existing solutions in a way that it combines a number of varying data sources in order to provide a more wholesome view of the application's operation. Other solutions such as FG (Perez et al. 2015) provide only limited hardware data combined to a monitor that is used to record method calls. The need for combining large variety of data sources was also noted in a study done on the field of self-adapting software (Salehie, et al., 2009).

CloudWave (Bruneo, et al., 2014) combines varying data sources with advanced analytic tools. In comparison the .Maintain framework offers more flexibility to monitoring by offering probes as a part of programming library. The approach the framework takes was not found in the literature, existing solutions act as a platform where applications operate. The framework is attached to application's program code. This way maintainability is taken in to consideration from early stages of development, the approach suggested in a report by Gartner (Kyte, 2012). In addition to probes and analytics, the framework also offers a range of tools that support continuous application development. According to literature this type of solution offering a wide range of tools does not seem to exist. Another difference to existing solutions is the integration to continuous deployment process, which is expected to support development and deployment processes. FG (Perez et al. 2015) supports developers with data about length of each stage of deployment process, but more data would provide extra value to developers (Olsson, et al., 2014).

# 6 SUMMARY

The purpose of this thesis was to construct a roadmap for a .Maintain framework. The framework will support the actions performed after initial deployment of an application. The goal is to reduce production costs and to speed up deployment of new features. From the literature the central tasks of this phase were identified to be: developing new features, fixing and identification defects and revealing additional requirements. Researchers have developed various methods and philosophies for this phase of development. One of the central concepts is continuous deployment. In this practice developers deploy program code to production environment as often as possible using automated tools. Other significant practice is DevOps which emphasizes cooperation between development and operation. In the center is the utilization of operational data. The roadmap for the framework was constructed based on these concepts and ideas using design science approach.

The proposed Framework consists of two key parts: (1) Analyzer combines data from multiple sources, the data is then turned to valuable information which can be used to guide application development. Tools for combatting data deluge are provided. Data sources are: the deployment pipeline, probes and hardware and infrastructure. (2) Programming library which has three key components. First part are the probes which can be attached to application to measure its state. Probes can be used to measure various parameters including user population, response time and user interaction patterns. Second part is the simulator which can be used to replace parts of the application with mock objects, this method simplifies unit testing. Third part of the library are the methods for controlling user access to parts of application, which will enable testing new features with live customers.

The framework has potential for reduction of development costs by providing information that can be used to make more intelligent design decisions. When completed .Maintain framework would fill a gap, as there are no existing tools that would provide support for all aspects of development of applications after initial deployment.

# 7 REFERENCES

Abdul, F.A. & Fhang, M.C.S., 2012. Implementing Continuous Integration towards rapid application development. In *2012 International Conference on Innovation Management and Technology Research*. Available at: http://dx.doi.org/10.1109/icimtr.2012.6236372.

Agarwal, P. & Puneet, A., 2011. Continuous SCRUM. In *Proceedings of the 4th India Software Engineering Conference on - ISEC '11*. Available at: http://dx.doi.org/10.1145/1953355.1953362.

Alwardt, A.L. et al., 2009. A lean approach to designing for software testability. In *2009 IEEE AUTOTESTCON*. Available at: http://dx.doi.org/10.1109/autest.2009.5314039.

Bruneo, D. et al., 2014. CloudWave: Where adaptive cloud management meets DevOps. In *2014 IEEE Symposium on Computers and Communications (ISCC)*. Available at: http://dx.doi.org/10.1109/iscc.2014.6912638.

Caron, E. et al., 2011. Auto-Scaling, Load Balancing and Monitoring in Commercial and Open-Source Clouds. In *Cloud Computing*. pp. 301–323.

Claps, G.G., Svensson, R.B. & Aybüke, A., 2015. On the journey to continuous deployment: Technical and social challenges along the way. *Information and Software Technology*, 57, pp.21–31.

Cois, C.A., Joseph, Y. & Anne, C., 2014. Modern DevOps: Optimizing software development through effective system interactions. In *2014 IEEE International Professional Communication Conference (IPCC)*. Available at: http://dx.doi.org/10.1109/ipcc.2014.7020388.

Corona Erika And Filippo, 2013. A Review of Lean-Kanban Approaches in the Software Development. *WSEAS Transactions on Information Science and Applications*, (10.1 ), pp.1–13.

Dzamashvili Fogelström, N. et al., 2010. The impact of agile principles on market-driven software product development. *Journal of Software Maintenance and Evolution: Research and Practice*, 22(1), pp.53–80.

Ekstrand, M.D. & Ludwig, M., 2016. Dependency Injection with Static Analysis and Context-Aware Policy. *The Journal of Object Technology*, 15(1), p.1:1.

Feitelson, Dror G., Eitan Frachtenberg, and Kent L. Beck, Development and Deployment at Facebook.

Fitzgerald, B., Brian, F. & Klaas-Jan, S., 2014. Continuous software engineering and beyond: trends and challenges. In *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering - RCoSE 2014*. Available at: http://dx.doi.org/10.1145/2593812.2593813.

Fowler, F., 2006. Continuous Integration. *ThoughtWorks*.

Hammant, P., Picocontainer. Available at: http://picocontainer.com/introduction.html [Accessed November 22, 2016].

Highsmith, J. & Cockburn, A., 2001. Agile software development: the business of innovation. *Computer*, 34(9), pp.120–127.

Humble, J. & Farley, D., 2010. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation (Adobe Reader)*, Pearson Education.

Jeff Widman, Stella Y. Hua, Steven C. Ross, 2010. APPLYING LEAN PRINCIPLES IN SOFTWARE DEVELOPMENT PROCESS – A CASE STUDY. *Issues in Information Systems*.

Kohari, N., 2012. Ninject. Available at: http://www.ninject.org/ [Accessed November 22, 2016].

Kohavi, Ronny, Thomas Crook, Roger Longbotham, Brian Frasca, Randy Henne, Juan Lavista Ferres, and Tamir Melamed, 2009. Online experimentation at Microsoft. *Data*

*Mining Case Studies*, 11.

Kyte, A., 2012. Four Laws of Application Total Cost of Ownership. Available at: www.gartner.com/.

Neely, S., Steve, N. & Steve, S., 2013. Continuous Delivery? Easy! Just Change Everything (Well, Maybe It Is Not That Easy). In *2013 Agile Conference*. Available at: http://dx.doi.org/10.1109/agile.2013.17.

Olsson, H.H. & Bosch, J., 2014. Climbing the “Stairway to Heaven”: Evolving From Agile Development to Continuous Deployment of Software. In *Continuous Software Engineering*. pp. 15–27.

Parekh, J. et al., 2006. Retrofitting Autonomic Capabilities onto Legacy Systems. *Cluster computing*, 9(2), pp.141–159.

Peffers, K. et al., 2007. A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, 24(3), pp.45–77.

Perez, J.F., Weikun, W. & Giuliano, C., 2015. Towards a DevOps Approach for Software Quality Engineering. In *Proceedings of the 2015 Workshop on Challenges in Performance Methods for Software Development - WOSP ’15*. Available at: http://dx.doi.org/10.1145/2693561.2693564.

Pulkkinen, 2013. Continuous Deployment of Software. *Cloud-Based Software Engineering*, 46.

Rahman, A.A.U. et al., 2015. Synthesizing Continuous Deployment Practices Used in Software Development. In *2015 Agile Conference*. Available at: http://dx.doi.org/10.1109/agile.2015.12.

Robillard, P.N., Kruchten, P. & D’Astous, P., 2003. *Software engineering process with the UPEDU*, Addison-Wesley.

Roche, J. & James, R., 2013. Adopting DevOps practices in quality assurance.

*Communications of the ACM*, 56(11), pp.38–43.

Salehie, M. & Tahvildari, L., 2009. Self-adaptive software. *ACM Transactions on Autonomous and Adaptive Systems*, 4(2), pp.1–42.

Savchenko, D.I., Radchenko, G.I. & Taipale, O., 2015. Microservices validation: Mjolnirr platform case study. In *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. Available at: http://dx.doi.org/10.1109/mipro.2015.7160271.

Shang, W. & Weiyi, S., 2012. Bridging the divide between software developers and operators using logs. In *2012 34th International Conference on Software Engineering (ICSE)*. Available at: http://dx.doi.org/10.1109/icse.2012.6227031.

Stolberg, S. & Sean, S., 2009. Enabling Agile Testing through Continuous Integration. In *2009 Agile Conference*. Available at: http://dx.doi.org/10.1109/agile.2009.16.

Von Alan, R.H. , March, S.T., Park, J. and Ram, S., 2004. Design science in information systems research. *The Mississippi quarterly*, 28(1), pp.75–105.

Wettinger, J. et al., 2014. Standards-Based DevOps Automation and Integration Using TOSCA. In *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*. Available at: http://dx.doi.org/10.1109/ucc.2014.14.