CONTINUOUS DEPLOYMENT FOR COMPLEX SOFTWARE INTENSIVE INDUSTRIAL SYSTEMS

Zulqarnain Haider
zhr15002@student.mdh.se

Examiner: Radu Dobrin
Mälardalen University, Västerås, Sweden

Supervisor: Antonio Cicchetti
Mälardalen University, Västerås, Sweden

Company Supervisor: Markus Lindgren,
ABB AB, Västerås, Sweden

September 6, 2017
Abstract

Processes to develop and deliver software have been evolved over the years. One of the primary motivations of this evolution, is gaining the benefits of shorter time-to-market. Continuous deployment is a recent trend to deploy software to the customers automatically and in continuous fashion. Organizations adopting this trend could reach the customers faster through quick deliveries and improve the quality and productivity of the delivered product by an early feedback, and hence achieve increased customer satisfaction. Complex software intensive industrial systems are large-scale, distributed over heterogeneous platforms and interact with several sensors and actuators. Enabling continuous deployment for these industrial systems needs a stable deployment process able to cope with domain specific requirements and challenges. Notably, the required quality attributes of the deployed software product as well as the challenges introduced by the customer-specific nature of the domain. In this thesis, we formalize continuous deployment for industrial systems by identifying the main factors of an appropriate deployment process. In particular, we investigate high-level requirements, required quality attributes of the software product, and challenges in the deployment. Based on this, we propose a continuous deployment pipeline and a set of activities incorporated in the stages of the pipeline, in particular deployment and post-deployment stages. Moreover, we suggest automation support for the activities to both shorten the delivery time and to preserve repeatability and reliability of the deployment process. The aim of such a process is to maintain the quality attributes of the deployed software. We perform a case study to validate the proposed model by implementing a prototype in an industrial system.
Acknowledgments

First, I would like to express my gratitude towards the company supervisor Dr. Markus Lindgren for his endless support and guidance throughout the journey of this thesis. I would also like to thank my academic supervisor Assoc. Professor Antonio Cicchetti for all fruitful discussions, valuable suggestions and always being there to support. I am also thankful of ABB Force Measurement for providing me this opportunity. Finally, I thank my family and friends for their moral support and encouragement.
# Table of Contents

1 Introduction 5  
1.1 Problem Formulation 5  
1.2 Thesis Outline 6  

2 Background and Related Work 7  
2.1 Overview of Continuous Deployment 7  
2.2 Technical & Business Characteristics of Continuous Deployment 10  
2.3 Related Work 12  

3 Research Method 14  
3.1 Threats to Validity 15  

4 Continuous Deployment in Industrial Systems 16  
4.1 High-Level Requirements in Industrial Systems 16  
4.2 Quality Attributes in Industrial Systems 17  
4.3 Continuous Deployment Pipeline in Industrial Systems 17  
4.4 Required Activities in Continuous Deployment Pipeline for Industrial Systems 19  

5 Case Study 21  
5.1 Rules-based Automated Deployment 22  

6 Discussions and Limitations 23  

7 Conclusion 25  

References 27
## List of Figures

2. A typical DevOps cycle representing the development and operations activities in gray and white phases respectively. ......................................................... 9
4. Architecture of hypervisor and container based virtualization. .............................................. 12
5. Flow of the research method followed in this thesis. ................................................................. 14
6. An example of heterogeneous platforms in industrial applications [3]. ................................... 16
7. Stages of the continuous deployment pipeline in industrial systems. ........................................ 17
8. Stages of the continuous deployment pipeline in industrial systems incorporating the organizational factors. ......................................................................................... 19
9. Stages of the continuous deployment pipeline mapping the deployment challenges in industrial systems. ....................................................................................... 19
10. Activities and their relationship to the quality attributes of the software product. .............. 20
11. Architecture of the proposed continuous deployment solution. ............................................... 22
1 Introduction

Companies selling products and/or services constantly try to reduce time to market and increase the frequency of product/service introductions, simply because historically it has often been the company first to market that becomes the most profitable. This is of course also true for companies developing and selling products containing software (to various degrees).

For companies developing software products continuous deployment is one important contributor to reducing time-to-market and increase release frequency, which allows software to be deployed to customers in a continuous fashion. In addition, continuous deployment conforms to the two important agile principles [4], i) “our highest priority is to satisfy the customer through early and continuous delivery of valuable software” and, ii) “deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale”. Many successful companies, such as Facebook [5], Netflix [6] and Github [7] are practicing continuous deployment, to deploy their systems to production. There are also studies, such as [8], that have identified that continuous deployment can lead to an improvement in quality and productivity, faster feedback from product use, quick deployments and thereby be an important contributor to increased customer satisfaction. Today it is mainly web companies that are early adopters of continuous deployment, where typically the main functionality of the product/services are deployed on servers controlled by the company, with often limited interaction with client side software. Potentially there can be many other kinds of companies, besides web companies, that could benefit from using continuous deployment.

This thesis focus on complex software intensive industrial systems, which in contrast to web software, frequently interact with several physical sensors and actuators, including often interaction with other systems as well. Examples of these kind are automotive, avionics, robotics, and process control systems. These systems are often large-scale, heterogeneous and distributed embedded systems. In this thesis we investigate the potential of continuous deployment in such industrial systems, to reap the benefits already achieved by web companies. In particular, we characterize the continuous deployment process in industrial systems by identifying the factors driving and impacting the design of such a process. The identified factors include high-level requirements, quality attributes of software product (also referred to as software product properties in this thesis) and the deployment challenges in industrial systems. Based on these unique requirements and challenges, we proposed different stages of the continuous deployment pipeline and a set of technical activities and their automation to produce a consistent process. Such repeatable and reliable process both enables faster deliveries and preserves the quality attributes of software product. To identify these factors and incorporate into a stable deployment process, we performed a literature and a case study in an industrial system. The result of the case study is a prototype presenting the implementation of a solution to a partial problem and validation of the feasibility of the proposed descriptive model of continuous deployment in industrial systems. To the best of our knowledge, the work presented in this thesis is a first attempt to characterize continuous deployment for complex software intensive industrial systems.

The thesis is focused on the release phase of a software development life cycle. We limit the scope by assuming that the organizations of the interest already follow agile and continuous integration practices to build their products. Therefore, the identified activities are only limited to the deployment of the software in the customer environment. Activities related to pre-deployment phases of the software may also contribute in faster and early deployments with an impact on the quality attributes of the software product, but such activities are out of the scope of this thesis. However, the proposed continuous deployment pipeline incorporates all possible stages of the deployment for the sake of completeness and also briefly discusses the challenges in other stages.

1.1 Problem Formulation

Continuous deployment enables earlier response to changes and faster maintenance by speeding up both the processes of development and delivery through automation. In order to be effective, continuous deployment requires to be immersed in an appropriate development process, including continuous architecture, continuous integration, continuous testing and continuous delivery [9]. The continuous development process should be focused on non-functional requirements or quality
attributes rather than the conventional functional ones. Since, it is crucial for the organizations to deploy quality software to stay competitive in the market. For example, an industrial plant down for maintenance directly impacts revenue due to stoppage in production. In such case contributing to maintainability of the software could reduce the down time, while more reliable software could result in less unplanned maintenance. In addition to this, customer specific nature of industrial systems introduce unique requirements and challenges which should also be incorporated in the process.

Based on the above mentioned requirements and challenges we propose following research questions:

- **RQ1: What are the characteristics of continuous deployment in industrial systems?**
  - RQ1.1: What are the challenges in employing continuous deployment in industrial systems?
  - RQ1.2: What are the required quality attributes in industrial systems?
  - RQ1.3: Which activities should be integrated in deployment process to enable continuous deployment in industrial systems?

- **RQ2: Is it feasible to realize continuous deployment for industrial systems, while still fulfilling required quality attributes?**

### 1.2 Thesis Outline

Remainder of the thesis is organized as follows:

- Section 2, presents the background and related work.
- Section 3, describes the research methodology followed in this work and threats to validity.
- Section 4, presents the continuous deployment in industrial systems.
- Section 5, describes a case study and implementation of prototype for an industrial system.
- Section 6, presents discussions and limitations of the work.
- Section 7, concludes the work with a summary and possible future directions.
2 Background and Related Work

In this section we first present the overview of continuous deployment, describing the processes and activities supporting continuous deployment. Next, we discuss the technical and business characteristics of the continuous deployment. Finally, we present the related work on continuous deployment found in the literature to distinguish our contributions.

2.1 Overview of Continuous Deployment

Processes of developing and handing over software to consumer for its intended utilization has been evolved dramatically. Traditionally, software have been developed and delivered using stage-gate models. Where in such models, typically, a software development life cycle consisted of sequential activities such as collecting requirements, developing and finally deploying software. With the emergence of agile software development in 2001 the approaches to develop software has been revolutionized. Such approaches utilize the concepts of iterative development with short cycles to enable early and fast delivery of software and achieve customer satisfaction. These approaches have been developed to support the Agile Manifesto [4] and twelve principle of agile software [4]. However, such practices are more focused on the team management for enabling short cycles and early delivery & feedback, for example Scrum and Extreme Programming (XP).

Olsson et al. in [1] identified a pattern in the evolution of software processes by performing multiple case studies. Figure 1 illustrates the evolution path with subsequent phases. The authors named this evolution stairway to heaven and discovered five phases of the organizations opting for continuous deployment of software. These five phases are, i) traditional development, ii) agile R&D organization, iii) continuous integration, iv) continuous deployment and, v) R&D as an experiment system. It has been identified that the first step towards moving to next phase is organizations maturity and capability of practicing fully the current phase. This implies that for an organization to achieve continuous deployment, two important pre-requisites are practices of agile and continuous integration. Next, we provide a definition of related concepts, not necessarily in the order of the above mentioned pattern.

Traditional development refers to the approaches which focus on the development of software in sequential manner. In such approaches, requirements are gathered early in a typical yearly based product development life cycle, for example Waterfall model. In later steps, software is designed, developed and deployed. The inherent issue with such methods is the lack of customer feedback and their integration with the product development process.

Agile development refers to the methods, where software is developed incrementally within short iterations along with an emphasis on empirical feedback for example Scrum. Such methods are focused on improving the team management and process, through a continuous self-organization and customer feedback. However, such agile methods could not guarantee customer feedback in short intervals. This is mainly due to the lack of incorporation of the production environment in the iterative loop, and to managing and verifying the product utilizing traditional development models [1].

Continuous integration transforms product development and system validation to agile from traditional approaches and fill the gaps of agile development as mentioned earlier. Continuous integration as defined in [10] is “a method of finding software issues as early as possible within the development cycle and ensuring all parts of the overall platform talk to each other correctly”. This
implies that it is a set of practices which motivates members of a development team to integrate their work frequently, and verify it utilizing automated builds and tests. As defined by Martin Fowler in his article\footnote{www.martinfowler.com/articles/continuousIntegration.html}, frequently refers to the daily basis; while enabling such frequent integrations minimize the time required for an idea to be nurtured as a product \cite{11}.

**Continuous delivery** enables delivery of the software to consumer or production in a continuous fashion, best defined in Martin Fowler words\footnote{www.martinfowler.com/bliki/ContinuousDelivery.html} as “a software development discipline where you build software in such a way that the software can be released to production at any time”. The main benefit of employing continuous delivery is reducing the risk of deployment by maintaining and promising a deployable software incorporating recent features, all the time.

**Continuous deployment** corresponds to the process of deploying the deliverable software to customer in continuous fashion. In \cite{2} continuous deployment is defined as a process of releasing software automatically to customer, once it passes all the automated tests. Whereas, per Martin Fowler, continuous deployment means automatically releasing the software into production many times a day. Organizations are deploying software from two times \cite{5} to as often as thirty times \cite{12} a day. Continuous deployment process includes different practices, tools, activities and technologies through which organizations could deploy new software, updates or upgrades continuously to production. In \cite{2} and \cite{9} the authors suggest that in order to introduce these activities in the deployment pipeline, organizations should have already been practicing agile and continuous integration process. In \cite{13} the authors identified 11 different activities, practiced by 19 different organizations to enable continuous deployment of the software. One of these 19 organizations deploys both web and desktop software; whereas, the rest only deploy web software. The identified activities by the authors are i) automated deployment, ii) automated testing, iii) code review, iv) dark launching, v) end-user communication, vi) feature flag, vii) intercommunication, viii) monitoring, ix) repository use, x) shepherding changes, and xi) staging. Three of the activities are of most importance and are practiced by all of the organizations i.e., automated deployment, automated testing and repository use. Additionally, a systematic mapping \cite{14} of continuous deployment literature discovered 10 different traits of continuous deployment as the theme of the literature. These traits are i) fast and frequent release, ii) flexible product design and architecture, iii) continuous testing and quality assurance, iv) automation, v) configuration management, vi) customer involvement, vii) continuous and rapid experimentation, viii) post deployment activities, ix) agile and lean, and x) organizational factors. The automation theme suggests the automation of the activities spanning at all the phases of the software life cycle i.e., from building to delivering and maintaining the software in production.

**Automated deployment** refers to the automation of all the steps required for delivering and executing the software in customer environment. In \cite{15} the authors characterized software deployment with the following set of activities:

- **Release** activity serves as a border between development and deployment process and refers to the packaging of software for transferring to production. Thus, it includes the practices of determining the resources and information required for deployment to consumer along with the releasing of the software package. The package contains system components and external dependencies. Additionally, advertising process is carried out to inform the stakeholders about the release.

- **Install** is a process to transfer and configure software into customer environment.

- **Transfer** refers to the activity where software is ported and the system is prepared for activation at customer machine utilizing the configuration.

- **Activate** refers to a set of activities responsible for installing and activation of all the applications and resources required to execute software along with actually starting the execution of software.

- **Deactivate** is a reverse process of activate and is responsible to stop any executing components, mainly to update the software.
- **Update** process is carried out to install new version of software, which might be released to introduce new features or improve efficiency, etc. It is different than install activity in a sense, that many dependent resources and applications are already acquired.

- **Adapt** is also a process of modifying software, however, it is carried out to incorporate changes introduced by local events and environment at customer location.

- **De-install** is inverse to install and involves deactivation and removal of software completely from the customer machine.

- **De-release** refers to retiring software and advertising to stakeholders that software is no longer supported by its producer.

Configuration management corresponds to different concepts in different contexts and domains. In cloud and DevOps configuration management server refers to the software for configuring the server and defining infrastructure as a code e.g., Chef and Puppet etc. Whereas, software configuration management is defined as tracking the changes in the software code utilizing tools like Subversion (SVN) and Git [14]. In the context of Component Based Software Engineering (CBSE), configuration management refers to the management of the configuration files which defines the assembly of different components for developing a product. In this thesis work, hereafter, configuration management refers to the management of configurations of component based software. Configuration files are script files written in domain specific language and defines how different components should be assembled to fulfill the requirements of application.

Delivering software to a customer and measuring the added value, requires many activities to be performed prior and after the software delivery. Such a life cycle which incorporates a feedback from customer/production environment introduces gains at different levels. For example measuring the performance of software could drive the innovation, earlier feedback could result in early fixes and thus customer satisfaction. In stairway to heaven [1], organizations practicing such a process is referred to as R&D as an experiment organization. DevOps is another related process, and has been defined in [10] as “a way of working that encourages the Development and Operations teams to work together in a highly collaborative way towards the same goal”. Figure 2 shows DevOps cycle of software delivery, where a close collaboration is depicted between development and operations. The phases on the left hand side depicts the development activities whereas, activities related to operations are illustrated on the right hand side.

![DevOps cycle](image)

Figure 2: A typical DevOps cycle representing the development and operations activities in gray and white phases respectively.

Activities contributing to continuous deployment such as automated deployment and configuration management are usually carried out during the Release phase of DevOps or any other software development life cycle. Such activities may guarantee reliable and repeatable process which is crucial to achieve the continuity. However, despite achieving such an automation, any
delay in execution of previous phases could compromise the continuous deployment. Hence, a dependency on the efficient execution of the Development phase. The development phase has many activities which are required to guarantee the effectiveness of the software and development process itself. For example coding, compiling and packaging along with the testing of the application at different levels. Such activities are performed at different stages of the development process and is often referred to as a deployment pipeline [2]. According to Martin Fowler[3] “deployment pipeline is a central part of continuous delivery”. In a deployment pipeline next stage is usually triggered once the previous stage has been executed successfully, and each stage addresses different aspects of developing and testing of a software. In [2] the authors, proposed a simple deployment pipeline with different stages as shown in Figure 3. However, the authors suggested that the activities and stages are subject to change depending upon the organization and industrial domain. Generally, in commit stage the code is compiled, unit tests are run and the software is packaged and stored in an artifact repository. The acceptance stage thoroughly tests the application using the test cases, which requires longer execution time and are often referred to as acceptance tests. In third stage the application is independently deployed to different environments for user acceptance test and capacity testing in parallel to the production.

![Figure 3: A typical deployment pipeline [2]](image)

2.2 Technical & Business Characteristics of Continuous Deployment

The journey of transforming an idea into a market-ready software service/product has been evolved during the last couple of decades. For companies, the evolution is not only on process of development but also the business models and economy. One such process focused evolution has been discussed in previous section. Transition in the process and invention of new technologies also impact the business models practiced by companies and vice versa. For example, transition to service from product based economy enables a quicker access to the market and continuous fashion innovation. Such transition could only be achieved through advancement in the processes and technologies, thus both the business strategy and technological advancement complement each other. Moreover, the benefits are two-fold, adding value to the economy of organization by adopting well suited business models and reducing the cost of developing and delivering the innovation, through technological advancements.

Web companies are employing cloud technologies to enable continuous deployment e.g., Facebook utilizes cloud to maintain & control the infrastructure and roll back a release as soon as a problem is discovered [5]. Formally, Cloud refers to as an internet accessible infrastructure, which provides on demand access to scalable and configurable computing resources such as servers, storages and applications etc., by considering service providing interactions. Internet of Things (IoT) extends the concept of cloud computing beyond computing and communication to include everything e.g., physical devices. In particular, IoT aims at interconnecting all objects (ranging from every day to industrial systems) capable of acquiring an Internet Protocol (IP), by providing an adaptive and self-configuring system which contains networks of sensors and smart devices. The objective of such connectivity is to make these objects intelligent, flexible, programmable and more interactive with humans [16].

Companies producing complex software intensive industrial systems could also benefit from cloud computing to reach the market early and add value to their business. For example, incorporating cloud and IoT, often together referred to as CloudIoT or IoTCloud, such organizations could enable services to consumers and businesses. Thus, both reaching the market early and becoming more flexible and agile in creating new business and revenue models[17]. Also, in [18] many such organizations initiated to evaluate the operational benefits and values that could be created by adopting IoT [18]. For example, using IoT to connect people, machines and computers along with cloud based data analytics contribute in intelligent and interactive systems. Such systems disclose great potentials in different domains such as industrial domain, smart city domain and health well-being domain [19].

In industrial systems, enabling continuous deployment through these technologies could also complement the organizations to practice novel business models e.g., pay for performance where customers are charged based on the performance of delivered services. Since, software is the core providing the features and values which drive such industrial systems, improvements through changes in the software are major contributors in overall performance and reliability. Moreover, by adopting this feedback loop an organization could aim at the elimination of unplanned maintenance through predictive, focused, and planned maintenance. In fact, predictive maintenance (planned) may reduce the cost if compared to unplanned maintenance, which in general is of expensive nature and endangers systems availability, which in turn affects negatively the continuous uptime.

Virtualization is the key underlying technology, which enables cloud computing to deliver resources as a service. The main advantages of virtualization are consistent, isolated, secure and independent hardware environments, and increase scalability [20]. Virtualization technologies can be classified in i) Hypervisor based Virtualization, and ii) Container based Virtualization. Hypervisor refers to a software technology that abstracts the underlying hardware, thus allowing many operating systems to run simultaneously on the same hardware. Each of these operating systems are running inside an emulator of the hardware often called virtual machines. Hypervisor manages these virtual machines and acts as a bridge between the real hardware and the emulated machines. The Operating System (OS) running inside a virtual machine is referred to as guest OS. Hypervisors can be classified as type 1 or type 2. The former is often called bare metal and runs directly on the hardware, whereas the latter requires an operating system (often called host OS) to manage the virtual machines. There exists many open source and commercial solutions, for example Xen and Hyper-V belonging to type 1 whereas, Kernel-based Virtual Machine (KVM), VirtualBox and VMware representing the type 2 [20]. Figure 4 represents the layered architecture of type 1 & 2 hypervisor.

Containers are light weight virtualization to provide an isolated environment for a process. The software for managing containers run on a host operating system and utilizes its functionalities to enable this isolation. Containers does not emulate the hardware, rather directly communicate with the host operating system for resource utilization. [20] Docker is an open source container management platform which provides both Linux and windows based containers. Docker supports application packaging along with its dependencies and refers to as docker image. Any running instance of this image file is a docker container running the application in isolated environment. There are many advantages of using such approach for packaging and running applications. For example, different versions of applications could be run side by side utilizing multiple containers for evaluation. Since the application executes inside a container, different environments like development, pre-production etc., does not effect the execution of application (theoretically). Moreover,
it provides a support of roll-back to previous version of application with ease. Figure 4 presents a comparison of the layered architecture of type 1 & 2 hypervisor and docker based container.

2.3 Related Work

In [1], Olsson et al. presented multiple case studies to identify the issues in transitioning towards a continuous deployment. In their stairway to heaven path they identified that continuous integration is a critical step towards achieving continuous deployment. The authors have identified three main challenges in a transition from continuous integration to continuous deployment. These challenges are i) complex configurations of software, ii) long internal verification loops and, iii) deployments being not transparent.

An extension to this work is presented in [21], where the authors have performed a multiple case study of five companies. Building upon same stairway to heaven path, the authors concluded two main activities to enable continuous deployment once continuous integration is established. These activities are i) architecture design, to support a rollback mechanism in case of unsuccessful deployment and, ii) deployment strategies, to enable deployment of modules and components in comparison to all software. Additionally, three organizational level activities have also been suggested, which are identification of lead customer, modification of business model to support continuous deployment, and alignments of teams in an organization such as release and R&D teams.

In [22], a further extension has been proposed to identify the practices and challenges in moving towards an innovation experiment system. The authors confirm that a candidate company has achieved continuous deployment of the product at team level. During this migration, the company faced challenges in shortening the complex and expensive verification along with validation cycle, and modifying the architecture to support independent deployments. However, the methods and practices to achieve such migration have not been discussed in the work.

Another multiple case study which is focused on adoption of DevOps has been carried out in [11], to study four candidate companies. one of the candidate companies produces industrial
automation systems, which is same domain considered in this thesis work as a case study. In the study four main challenges have been identified which are, i) different configurations and limited visibility of customer environment, ii) difficulty in automating the deployment due to complex hardware dependencies and compatibility of multiple versions of software, iii) lack in technology to deploy features and updates in different customer specific environments and, iv) enabling feedback from production environment.

Above mentioned studies identifies a need of automated deployment and configuration management framework to enable continuous deployment. Many researchers started to evaluate the cloud based deployment and management approaches to introduce potential deployment environments, for example virtual machines and containers. In [23], a sand-boxed environment (Docker containers) has been evaluated to deploy the software in automotive industry. In such a domain, the software has safety and real-time requirements which also contributes in the complexity. The authors identified that such sandbox environments does not affect the performance of the deployed software or cause any overhead in comparison to native deployment environment. The authors also suggested that microservices embedded in such containers could be utilized in cyber physical systems domain.

In [20], Ramalho et al. evaluated docker containers and virtual machines by executing several synthetic benchmarks to measure the introduced performance overhead. The authors have observed that the containers produced promising results in comparison to virtual machines, which introduced significant overhead. The software has been targeting different network edge devices to initially process the data.

Felter et al., in [24] also performed a comparison of virtual machine and docker environment. The authors observed that containers yielded equal or better performance compare to virtual machines in almost all cases. The authors also discovered that careful tuning of containers is required to support input/output intensive applications.

In [25] the author, proposed an open source infrastructure incorporating the technologies like VirtualBox, Docker and Jenkins to enable continuous deployment in vehicle domain. The author observed that maintenance and updates effort is very low due to having a high degree of automation.

In [3] the author, motivated and described initial research in the direction of deployment on heterogeneous platform. More specifically, the focus of the research is software deployment on heterogeneous platform while preserving extra functional properties such as performance etc.
3 Research Method

In this section we present the research method, adopted in this thesis work, and threats to validity.

The initial objective was to conduct a state-of-the-art literature survey of continuous deployment to identify the best practices in complex software intensive industrial systems. But, during our initial attempt to establish the related work we found a recently published systematic mapping of continuous deployment literature [14], and also discovered that the area was open with a number of challenges. Therefore, the focus has been changed to proposing a solution to the problem. We have adopted the research method presented in [26] to propose the solution. The steps of the research process are outlined in Figure 5.

![Figure 5: Flow of the research method followed in this thesis.](image)

During the literature review, we have identified several qualitative studies discussing open challenges [11] [22] [21] [1], implementation [25] and quantitative analysis [23] [20] [24] of solutions to partial problems in different complex software intensive industrial domains and solutions in web domain [5] [6] [7]. In addition to this, we also identified that these solutions are highly customized and serve the purpose of specific organization and domain. Therefore, we started with characterizing the continuous deployment in complex software intensive industrial systems. The aim was to identify what are those factors, which needs to be incorporated in the context when designing such a solution for industrial systems. This led us to propose a descriptive model of continuous deployment in industrial systems. We performed a case study to validate the feasibility of the proposed characterization and based on the peer reviews and literature study we iteratively improved our solution. After several iterations, we formalized the characterization of continuous deployment in industrial systems with factors like high-level requirements, quality attributes of the software product and deployment challenges in industrial systems. These factors are incorporated in proposing the continuous deployment pipeline for industrial systems and a set of activities are
presented which are required to be automated and incorporated in the stages of the pipeline to enable continuous deployment. In addition to validating the characterization of continuous deployment in industrial systems, the case study presents an implementation of a prototype in industrial system to demonstrate the feasibility of the proposed approach.

### 3.1 Threats to Validity

To compile and analyze related works, only one database i.e., Scopus is used to identify the state-of-the-art and related studies published after year 2014. Since, the studies mentioned in the related work and published before year 2014 are identified from a recent systematic mapping of continuous deployment of software intensive products and services [14], we feel adequate coverage was achieved. The citations of the identified literature have been further explored using Google Scholar to find additional related work. This partial Snow-balling search, which refers to the study of citations and references of the related studies, increased the coverage of identifying the related literature. Due to the restricted search method employed, there is a likelihood that we may have omitted certain results in our analysis. This would have a bearing on the challenges addressed in our case study.

Due to the limitation of time, the implementation of the proposed continuous deployment architecture has not been carried out on the actual Stressometer industrial system, rather a solution to the partial problem is implemented on off-the-shelf hardware. Such an implementation could enable to measure the metrics like reliability, availability etc. of the delivered software product along with the delivery time. These metrics could be utilized to analyze and evaluate the proposed solution which would help in building greater confidence in the validity of the results.

The case study targets a single industrial system i.e., Stressometer. However, the identified requirements and challenges in our case study are similar to those mentioned in the literature, thus covering a wider range of industrial systems. The generic approach adopted in our case study, though representative of a majority of the industrial systems deployed, could exclude certain specialized systems which would need further extensions to the solution.
4 Continuous Deployment in Industrial Systems

In this section, we present a formalization of the continuous deployment in industrial systems. First, we discuss the high-level requirements and quality attributes required in industrial systems. Next, we present a continuous deployment pipeline representing different stages of the deployment. Each stage incorporates different activities to deliver quality software to the customers depending upon the requirements. Automation of these activities may enable the faster delivery but also impact the quality attributes both in positive and negative manner. The unique customer specific nature of the industrial systems impact the automation of these activities and introduce several different challenges, which are mapped to the stages of the deployment pipeline. Finally, we suggest the activities required to be automated and immersed in the deployment pipeline to enable continuous deployment and their impact on the identified quality attributes of the software product. The suggested activities belongs to the final two stages of the proposed deployment pipeline i.e., Customer acceptance stage and Performance feedback stage.

4.1 High-Level Requirements in Industrial Systems

Industrial systems has unique high-level requirements, identified by studying the case as described in Section 5 and literature [11] [22] [21] [1].

- **Various Configurations of Software**: when addressing the requirements of a diverse customers set, developing a software with a static behavior could not suffice all the contexts. Instead, organizations design the software with configurable behavior. Developing such a software may reduce the cost and could possibly be configured to provide diverse functionalities to different customers. Additionally, this approach may also support the third party systems customers have for other functionalities, with ease. However, this comes with a cost of management of these configuration artifacts for each customer. This also add constraints on testing the software product, since customer specific solution results in less generic solutions. Due to these, the customers environments are often less transparent and very costly to re-produce at factory for testing.

- **Heterogeneous Platform**: Industrial software usually consists of different applications, deployed on heterogeneous platforms and performing different tasks. Figure 6 depicts two different hardware platform that is a Field-Programmable Gate Array (FPGA) and a Central Processing Unit (CPU) having different software platform, in a typical control process.

- **Deployment strategies**: refers to the policies for deploying the software to customers. A seamless updating or updating the system without notifying the customer, could jeopardize the system’s availability. On the other hand, notification to the customers and requiring their approval for updating the system might compromise the continuous aspect of the deployment. A trade-off is required on incorporating the customer in deployment process to guarantee the required quality.

- **Feedback Driven Innovation**: refers to driving the course of innovation based on the feedback received from the customer. This is required to measure the added value and support continuous deployment of feedback driven innovation.

![Figure 6: An example of heterogeneous platforms in industrial applications [3].](image-url)
4.2 Quality Attributes in Industrial Systems

In industrial systems following quality attributes of the software product are of high importance, which are identified by studying the case as described in Section 5 and literature [11] [22] [21] [1].

- **Reliability**, is a property of a system which states that system performs its intended functionalities accurately and as per its specifications for a certain stated time period.

- **Availability**, is the ability of a system to continuously provide reliable services.

- **Maintainability**, ensures that system could undergo maintenance with ease.

- **Performance**, refers to the response time of a system to an event or the number of transactions processed in a unit time.

The consequences of failure in maintaining these quality attributes could lead to several implications depending upon the nature of industrial system. For example, a robot functioning in an automotive assembly line; the failure of such robot due to compromised availability or reliability may cause the entire production line to stop and thus result in loss of production, delays and revenue loss etc. Whereas, compromised performance of such robot can result in less precise welding or bad welding joints. Which in turn has a negative impact on revenue and customer satisfaction. In case of the Stressometer control system, which measures and control the flatness of metal strips in cold rolling mills, compromised performance can result in bad flatness or strip breakage. This can increase the scrap and damage the actuators and eventually increase the losses in revenue and/or maintenance cost.

Due to this, the customers of such industrial systems often hesitate to upgrade the system unless a maintenance is needed. Therefore, the process of continuous deployment should be driven by, and maintaining, these quality attributes of the software product. Moreover, it should incorporate the fail safe means such as rolling back to stable version and audit trails of deployment to ease the maintenance.

4.3 Continuous Deployment Pipeline in Industrial Systems

In industrial systems the continuous deployment pipeline has different stages and activities, which are discussed next, in comparison to the typical deployment pipeline shown in Figure 3. The contrast is driven by the unique requirements mentioned in previous sections. Figure 7 illustrates the proposed stages of a continuous deployment pipeline.

![Diagram of the continuous deployment pipeline in industrial systems](image)

In commit stage various development teams commit code to the respective build server. The code could be an implementation of new feature or a bug fix. Each of the commit results in a fresh build of the respective module of the application. All these agile teams produce the software in various concurrent development pipelines. Each of these simultaneous pipeline follows a standard continuous integration cycle, where practices such as code review, static analysis, automated unit
and integration testing are performed depending upon the organization culture and required quality. Each of these verification activities increase the confidence on the latest build and is ready for next stages. The number of development teams could easily grow in the organizations where software is divided into different modules. Each of these modules could take different time from a code check-in to producing a stable build deliverable to next stage. This difference is mainly due to the hardware dependency of a certain module [11], as hardware development requires longer time or could be due to unit and integration testing taking longer computation time. A successful execution of this stage results in the compiled artifacts which are then stored in a central repository for upcoming stages.

The factory acceptance stage could be seen as acceptance stage [2] in a typical web companies software deployment pipeline. The key difference is the difficulties in producing production like environment and configuring that environment. In web companies, virtualization is commonly used to produce such production-like environment and an automatic configuration is carried out for acceptance testing. The simplicity in achieving this in web companies comes from the nature of software, i.e., dependency on off-the-shelf hardware and typically same configuration for all the customers. However, in our scenario the dependency on the specific test hardware or simulation software for producing a production-like environment may require manual intervention and slows down the execution of the activity. Once the production-like environment is produced and configured the latest build is fetched from the central repository and deployed in this newly configured environment. The application is verified and validated using regression and acceptance testing. An inherent issue, which is not unique to hardware dependent software is computation time required for executing a set of test cases. In contrast to web companies this is challenging and often impossible to produce many production-like environment through virtualization for parallel processing, mainly due to the hardware dependencies and thus the involved cost.

The Customer acceptance stage requires the application testing in customer specific requirement. The applications considered in this thesis work, has a large set of different customers using different third party systems, thus resulting in various different configurations. Unlike web companies these environments are often impossible to reproduce in-house at reasonable cost. Due to the problems of reproducing the customer environment in-house there may be a higher risk, when compared to web-companies, to experience unforeseen side-effects where configurable software is deployed to a multitude of different customer environment with different processes, sensors and actuators etc. Therefore, further adjustments to the process are required. For example, introducing a rollback mechanism to cope with the builds failing test cases. Also, a redundant hardware in many cases to not interrupt the already running instance of the application. These adjustments could contribute in the reliability and availability of the system. However, a lack in technology to deploy to these customized environment automatically could result in manual steps and often depends on domain expertise. Moreover, dealing customer concerns to update/upgrade the system as well require a manual intervention. For example customer willingness and manually choosing the customers to deploy the updates. A successful customer acceptance triggers the updated software to go live and take over the previous version.

The Performance feedback stage has a similar activity in a typical deployment pipeline [2], that is a continuous monitoring of the environment and measuring the feature usage. In industrial systems it is difficult to monitor customer environments and usually the feedback is conveyed and received manually by the customer [11]. In contrast, web companies analyze and measure performance through the feature usage to support the data driven development in many cases. However, the trend is changing fast in complex software intensive systems too, when there are services providing additional values to the customer.

In addition to the above mentioned stages of the deployment pipeline, the organizational culture also plays a vital role in enabling the continuous aspect of it. It has been identified in [2][9][1] that adoption of agile development culture is a prerequisite for achieving the continuous integration and deployment. Therefore, we incorporated an additional zero stage which refers to the adoption of agile culture in the deployment pipeline and is illustrated in Figure 8.

Enabling a continuous deployment pipeline, requires that all of the above mentioned stages are synchronized and progress with similar frequency. This is important to avoid bottlenecks, since, a stage requiring longer duration to execute affects the subsequent stages in a negative way and compromise the continuous aspect of deployment [2]. Thus, automation of the activities of these
Figure 8: Stages of the continuous deployment pipeline in industrial systems incorporating the organizational factors.

Figure 9: Stages of the continuous deployment pipeline mapping the deployment challenges in industrial systems.

4.4 Required Activities in Continuous Deployment Pipeline for Industrial Systems

In order to mitigate the challenges depicted in Figure 9, we propose a set of activities which a continuous deployment pipeline for industrial systems needs to implement in order to support the high-level requirements and maintain the properties mentioned in Section 4.1 and 4.2 respectively. Figure 10 relates the activities with the properties of the software product. These set of activities span on the release phase of software deployment and are as following:

- **Automated deployment**, enables the deployment of application at customer environment automatically as soon as a stable build is available. Such automation of the deployment guarantees a repeatable and free of manual steps process. Thus, may contribute in reliability, maintainability and availability of the software product.
Automated Rollback, supports rolling back the version of software as soon as an issue is discovered in the deployment of software. Thus, may contribute in reliability and availability of the software product.

Modular deployment of the software, supports the independent deployments of different modules of the software. This also encourages that the design of software should be module based, where different components could be deployed without affecting already deployed software. This could enable quicker maintenances, thus increases the maintainability and availability of the software product.

Visibility and Transparency in the process of deployment is required to visualize and manage the process. Views providing the diagnostics of the software and customer environment with a support of manually managing different aspects of the deployment may contribute in the reliability of the software product.

Deployment strategies refers to the schemes for deploying the software to different customers distributed at different geographical locations. The schemes could be driven by the customers interest in different kind of updates. This also incorporates the customers consent and preferences of deployment hours. Such schemes may contribute in the maintainability, availability and reliability of the software product.

Configuration Management, refers to the automated management of different configurations of software. Such software configurations provides different functionalities based on customer interest. Thus an automation could reduce the time to change a customer configuration with reliability, thus, contributing to reliability, availability and maintainability of the software product.

Monitoring Customer environment could enable measuring the performance of the software and complementing the feedback loop. The feedback drives the innovation and contributes in the maintainability of the software product.
5 Case Study

In this section we discuss the study of an industrial system and an implementation of the prototype of continuous deployment for the system. The aim of the study is to demonstrate the feasibility of continuous deployment in industrial systems. The study of the system has been carried out through reading different internal documents and several informal discussions with the industrial experts. This led us to identify the high-level requirements, software product’s properties and deployment challenges for the industrial system. Moreover, the discussions also led us understanding the architecture of the software product and studying the provided simulator of the industrial system enabled understanding the behavior of different components of the software. Next, we briefly describe the organization and industrial system case and present the architecture of the continuous deployment solution and implementation of the prototype.

ABB produces innovative solutions for a wide range of domains, such as robotics, power and industrial automation. In industrial automation, ABB is providing control and optimization solutions to improve the processes of oil & gas, chemicals, metals and materials industry etc. In this case study we have put focus on ABB’s Stressometer\(^4\) product, which is used for flatness measurement and control in mainly cold-rolling mills. The system offers optimal flatness performance, by adjusting the roll gap through mix of mechanical and thermal actuators within the mill to minimize the flatness error. The system provides following two main functionalities:

- **Flatness Measurement**: the flatness of metal strip is measured using Stressometer roll. The roll has several pressductor transducers which generates a signal in the presence of mechanical force due to the changes in the electromagnetic field. These signals can, when combined with other physical relationships and measured variables, be used in computing the flatness of the strip.

- **Flatness Control**: the control unit produces the command signals to the mechanical and thermal actuators of a mill and thereby indirectly forming a roll gap which achieves the desired flatness.

The software which is providing these functionalities has component based architecture, where different components are assembled to produce an application. The assembly of these components is described in configuration files written in a domain specific language. Customers are interested in different applications due to the different requirements. In addition to this, customers have different rolling mills manufactured by different vendors thus require a customized application. Therefore, for every customer the relevant components are assembled and their parameters are set to produce an application which fulfills their specific needs.

A high-level architecture supporting continuous deployment for Stressometer is proposed in Figure 11. Different development teams commit the code, which is build, packaged and tested according to the stages described in Section 4.3 and activities practiced in organization. Due to the limitations in this thesis, activities carried out in pre-deployment phases are not considered in the proposed architecture. The successful build is then stored in the "Build Repository" on cloud. In addition to this, Application Engineer commits the configuration files to "Configurations Repository" on cloud. These configuration files corresponds to customer specific application configuration. Continuous Deployment Application consists of following different services which automate the activities identified in Section 4:

- **Automated deployment Service** gets triggered and deploy the software product to all the customers as soon as a successful build is available in the build repository.

- **Roll back service** triggers a roll back to the previous stable version in case of issues.

- **Monitoring service** performs diagnostics on the data received from the customer’s environment to fulfill different requirements such as business, application health etc.

- **Configuration management service** manages the configurations of the customers.

- **Management Service** is used to manage the deployments manually by Operations Manager.

\(^4\)www.new.abb.com/products/measurement-products/flatness-measurement/
- Rules-based automated deployment service deploys the software product to the customers based on the predefined rules representing the deployment strategies customers opted for.

Moreover, a Customers Database in the cloud holds the data related to all the customers required by these services.

The proposed architecture is implemented as a prototype, which presents two of the above-mentioned services i.e., management service and rules-based automated deployment service.

Management service uses Portainer \(^5\) to enable the management of different factories. Portainer is an open source application for managing the docker containers remotely and fits very well in the proposed architecture. Operations manager could visualize and manage the customers environments, from deploying newer version to rolling back and resources acquired by the container running the software product e.g., memory usage and processor utilization etc.

### 5.1 Rules-based Automated Deployment

Rules-based automated deployment is a special case of automated deployment, where the deployment of the software product is driven by the preferences of the customers regarding the types of the updates. Each customer requests a deployment policy which is defined as the rules for deployment to that specific customer. To this end, the service defines three different policies i.e., bug fixes, minor and major. Architecture illustrated in Figure 11 shows factories on the right hand side (representing different customers). Each factory machine has Docker engine installed and has container based infrastructure as depicted in Figure 4. Docker engine consists of docker daemon, which is the docker server process responsible for managing docker containers. It also contains docker API and Command Line Interface to interact with docker daemon for management of containers remotely and locally respectively. The service communicates with docker daemon for deployment of the software using same docker API. The service runs in cloud and monitor the build repository. As soon as a new build is available in the repository, the application retrieves the customers and associated deployment policy from the database, connects and checks for the running version of the software product and then based on the retrieved rules deploy into the customers environments.

\(^5\)https://portainer.io/
6 Discussions and Limitations

This section presents a discussion on how the proposed research questions are answered, the findings of the thesis and limitations of the solution.

To the best of our knowledge, the work presented in this thesis is a first attempt to characterize continuous deployment for complex software intensive industrial systems. The formalization presented in 4 has been achieved by studying the literature and industrial system case and directly answers the research questions RQ1 and RQ1.1-1.3. The modeling of continuous deployment process as a continuous deployment pipeline separates the concerns and provide multiple viewpoints of the process. Moreover, incorporation of the impacting factors in the pipeline suggests automation of required activities. Thus, not only implementing the services to fulfill requirements and faster deployments but also maintaining the required software product’s properties. The limitation of presenting the characterization for organizations already practicing agile and continuous integration results in the identification of the activities belonging to the deployment and post-deployment stages. There could possibly be many required activities in other stages to enable faster deployments and also directly impacting the required quality attributes of the software product. For example, automation of activities such as integration testing, acceptance testing, smoke testing could significantly increase the speed of execution of respective stages and also impact the reliability of the software product.

The feasibility of continuous deployment for industrial systems represented in the research question RQ2 is determined by proposing the continuous deployment solution for Stressometer industrial case and implementing a prototype targeting partial problem. The proposed architecture could well fit in the recent trend of connecting such industrial systems to cloud to benefit from different services. This trend is driven to offer different services for enabling the required feedback loop and gain business value. The continuous deployment application also offers several required services deployed to cloud for enabling continuous deployment of the software product. Such practice could enable an organization to offer the produced industrial systems as a service to their customers. Thus, charging their customers based on the added value such as usage and performance and often referred to as pay per usage and pay for performance respectively. In such a case, monitoring service measuring the delivered performance in the customer’s environment and utilizing the feedback loop to drive the innovation and deliver it utilizing the proposed services (i.e., automated deployment service and rollback service etc.), to maintain/improve the delivered performance. In addition to drive the innovation, the monitoring service also computes the cost to be charged from the customer based on the delivered performance and usage.

The implemented rules-based automated deployment enables faster delivery along with an increased confidence on the software product due to the elimination of manual deployment steps. This may also encourage the customers to opt for new updates, which were avoided due to the manual steps and longer down time in past. Moreover, the option of choosing among different deployment strategies provide more flexibility to customers for updating their systems. Another important constraint in such deployment is the customers consent which could affect the continuous aspect of the deployment. Therefore, a parallel deployment of different versions of the software product and a smooth transition could be a key contributor to mitigate these issues.

The proposed architecture, is supported by container based infrastructure (Docker). To this end, docker provides support for windows and Linux operating systems. In our case, the software product requires windows environment to execute and perform its functionality. However, this particular system similar to other industrial systems is distributed over heterogeneous platforms. Such platforms e.g. FPGA and Real-Time Operating System (RTOS) etc. are not yet supported by docker technology. Another, inherent issue with windows based docker containers is the size of docker images. For example, a windows server image size is 9.5 GB which is huge in comparison with ubuntu image size i.e. only 188 MB. This could significantly affect the time to deploy the software product into the customers machines due to the larger size and also introduce implications in the handling of such large files. Linux has been using containers for almost a decade [20], whereas windows has not been designed in the way to support this technology. But, Microsoft has started to support the technology and efforts to reduce the image size is already on the way.

To this end, the proposed automated deployment services only support deployment of the complete software product. A modular deployment, which only allows the deployment of those
modules which offer changes/new functionalities, may contribute in smaller and faster deployment of updates to the customers. But, this would require refactoring and changes in the architecture of the software product to support such modular deployments.
7 Conclusion

In this section we conclude the thesis by presenting a summary and possible future directions. This thesis work explores continuous deployment in complex software intensive industrial systems. We proposed a descriptive model of continuous deployment for industrial systems, which identifies the factors impacting the continuous deployment. These factors include the high-level domain specific requirements, software product properties, continuous deployment pipeline, activities required for enabling continuous deployment and their impact on the software product’s properties. The proposed characterization, to the best of our knowledge, is a first attempt to formalize the continuous deployment for industrial systems. The characterization is validated through studying the literature and an industrial system case to iteratively improve the solution. In the end, we proposed an architecture of the continuous deployment solution for industrial systems. The solution consists of several required services enabling continuous deployment. In the prototype, we have implemented two services i.e., management service and rules-based automated deployment to demonstrate the feasibility of the approach. The former enables the visualization and transparency in the deployment process through manual management. Whereas, the latter deploys the software product into the customers environment automatically based on their pre-defined preferences for the type of updates i.e., bug fixes, minor and major updates.

The work presented in this thesis could be extended in the following ways. An implementation of the proposed architecture and measuring both the software product’s properties and important metrics such as time to deliver the software product to the customers. Such quantitative analysis could reveal the added value and reasons on how effective the solution is for such industrial systems. Since, these industrial systems are distributed over several platforms; investigation and implementation of the container based approach for platforms like RTOS and FPGA etc. would contribute in the completeness of the solution for industrial systems. Finally, incorporating the pre-deployment activities and factors in the proposed characterization could benefit the organizations still lacking the practices of agile and continuous integration. This also contributes in the measurement of the software product properties and delivery time to analyze and evaluate the overall process.
References


