Design Requirements to Improve Adoption of Continuous Development Services

by

Trevor Rae
B.Sc., McMaster University, 2018

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTERS OF SCIENCE

in the Department of Computer Science

Design Requirements to Improve Adoption of Continuous Development Services

by

Trevor Rae
B.Sc., McMaster University, 2018

Supervisory Committee

Dr. D. Damian, Supervisor
(Computer Science)

Dr. N. Ernst, Departmental Member
(Computer Science)

Supervisory Committee

Dr. D. Damian, Supervisor
(Computer Science)

Dr. N. Ernst, Departmental Member
(Computer Science)

ABSTRACT

The adoption of Continuous Development presents many challenges to users and organizations. The complexity of Continuous Development adoption is partially attributable to the diversity of the challenges faced, including technical challenges, cultural challenges, compliance regulations, and lack of understanding. In this thesis, I worked with industry partner IBM to improve their Continuous Delivery Pipeline offering to overcome adoption challenges faced by their users. Following Hevner's Three Cycle Design Science Methodology, my research had two distinct stages: **Characterizing Continuous Development Adoption Challenges** and **Creating Design Requirements to Aid Organizations Offering Continuous Development Services**. Both stages necessitated involvement from both academic literature and industry collaboration with IBM. Industry collaboration included interviews, surveys, developer forum analysis, and collaboration with IBM's "Continuous Delivery" teams. The design requirements I developed in this thesis addressed cultural, technical, compliance, and knowledge gap adoption challenges that were identified during the problem characterization stage. When tested within the Continuous Development community, feedback indicated that my design requirements would add value to users' development process, enabling their Continuous Development adoption. This thesis provides a foundation of empirical research for future study and a set of guidelines for industry practitioners.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

Daniela Damian, for opening the door and guiding me into the world of research.

Samantha Orr, for turning this experience into a journey.

My Family and Friends, for the never ending tirade of love and support.

Nancy Ami, providing continual support and guidance on my writing.

Mike Wilson, providing invaluable insight into the labyrinth of industrial software development

IBM CAS, for funding this research.

DEDICATION

Jim Marr

No matter how long the ride, how far the destination, how bumpy the road, you are always here by my side.

Chapter 1

Introduction

1.1 Motivation For Research

Adopting new technologies is often a daunting task. Continuous Development is no different. Continuous Development extends the principals of DevOps [Claps et al., 2015], which is the union of development and operation activities in software development practices. Continuous Development is the umbrella term encompassing emergent continuous practices [Fitzgerald and Stol, 2015], for which the emergent nomenclature is C*. Continuous practices are built to deliver value continuously via small incremental changes, guided by a continuous feedback loop, which is integral to its success. The goals of continuous practices are waste reduction and consistent feedback [Leppänen et al., 2015]. A common and well known continuous practice is Continuous Delivery. Continuous Delivery is a development method in which code changes are rapidly developed in small increments and with help from an automated pipeline delivered to end-users in short delivery windows, spanning from hours to days [Humble and Farley, 2010]. Organizations face a series of challenges when adopting continuous development practices. Providers of continuous development services, such as IBM, Google, Amazon, and Microsoft, are painfully aware of the difficulty their users face during the transition to continuous development practices. However, providers struggle to identify specific challenges and their solutions. The significance of this is compounded as global interest in Continuous Development continues to grow [Claps et al., 2015]. During the 5 years from 2010-2015 the Google search trend for "Continuous Delivery" has grown exponentially [Chen, 2017]. Despite the growth in popularity, the 2019 DORA report shows that the vast majority of

organizations are stuck in the middle of the assimilation process [Forsgren et al., 2019]. Most news about Continuous Development adoption and use is focused on large organizations such as Netflix[1] and Facebook[2]. Similarly, a large number of medium-sized organizations are also transitioning to Continuous Development [Leppänen et al., 2015]. Data indicates that smaller organizations have a higher success rate for Continuous Development adoption [Laukkanen et al., 2017]. The DORA report found that enterprise organizations (over 5000 employees) are out performed by smaller organizations [Forsgren et al., 2019].

Some Continuous Development challenges are known in the area of assimilation methodology, cultural resistances, and stakeholder conciliation. However, Continuous Development tools that are being created do not necessarily focus on alleviating these challenges. The primary motivation for this research is to help adopters of Continuous Development by providing improvements, validated by research, to continuous development services. To that end, I have partnered with IBM's Continuous Development services team to assist them in improving the adoption of their Continuous Development services with design guidelines validated by this research.

1.2 Research Questions

With the exploration and investigation of this topic, two research questions emerged:

RQ1: What are design requirements to improve the adoption of IBM's Continuous Development services?

RQ2: What challenges do adopters of continuous development practices face?

1.3 Methodology

The fast-paced development of the technology industry can make it challenging for traditional research methodologies to produce evidence-based best practices at the same pace. With industry constantly evolving, there's a risk that results of traditional research could be rendered obsolete by the time they reach publication. In an effort to deliver value to the industry, an iterative design science methodology was employed.

In order to achieve this, constant industry collaboration was required. IBM and SEGAL labs enjoy a long history of collaboration through industry-applicable

[1] https://blog.newrelic.com/technology/data-culture-survey-results-faster-deployment/
[2] https://www.infoq.com/presentations/Facebook-Release-Process/

research. I traveled to IBM and met a development team interested in taking a research-guided approach to improving their services. They were hoping to enable better decision-making regarding the direction of product development. The focus of this research project was to address and alleviate Continuous Development adoption challenges through incremental improvements to IBM's Continuous Development services. This thesis was done in two main stages. The first was problem characterization. Problem characterization consisted of a progressive series of investigations in the continuous development domain to identify and refine the challenges existing in continuous development adoption, each building on knowledge obtained from the previous investigations. These investigations included interviews and developer forum analysis. By speaking with the developers and users of IBM's Continuous Delivery Pipeline, along with users of other pipelines, I was able to refine the set of challenges that made it difficult to adopt IBM's pipeline. In the second stage of this thesis, the artifact I created was a set of design requirements to improve the adoption of IBM's Continuous Delivery Pipeline. The first version of the artifact was created very early in the research process. The initial design guidelines were whiteboard mock-ups, which evolved through feedback generated from problem characterization and consultation of existing literature. The artifact ended up as high fidelity walkthroughs, detailing the use, functionality, and display for each design requirement. For each iteration of the artifact, the design was validated through interviews, surveys, as well as close collaboration partner IBM. The validation was used to inform the next iteration of the artifact's design.

1.4 Contributions

This research expands upon the existing literature found in the continuous development domain. Industry specific challenges that users face when adopting continuous development services are investigated and documented. Self-reported empirical data on software development processes from both within IBM as well as the greater software development community is shared. The data generated in this thesis can be used in future research regarding continuous development adoption challenges and in exploring emergent continuous practices. Some correlations drawn in Chapter 5 may be of interest to researchers studying feedback collection patterns in software engineering. Chapter 4 presents an industry perspective on requirements engineering.

The design requirements artifact that I developed in this thesis has been assimilated

into the IBM Continuous Delivery team's development plans. These design requirements will be implemented into the Toolchain offering to assist users of IBM's Continuous Delivery Pipeline in their adoption and use. This should enable users to more easily overcome the barriers of adoption and improve their product development process through the use of continuous development practices. As this thesis is publicly available, it is also my hope other organizations developing tools and services to facilitate Continuous Development adoption can utilize this thesis to improve on their offerings.

1.5 Thesis Outline

Chapter 1 - Introduction introduces the thesis and outlines the research topic.
Chapter 2 - Background and Related Work provides a synopsis of relevant work being conducted in Continuous Development research. Additionally, this chapter details key terminology used in this thesis and describes this history of Continuous Development.
Chapter 3 - Methodology discusses the methods used in the design science methodology.
Chapter 4 - Problem Characterization describes the results of this research methodological step to characterize the problems addressed by this thesis.
Chapter 5 - Artifact Design and Evaluation details the steps taken to iteratively create and evaluate a set of design requirements to improve the adoption of Continuous Development services.
Chapter 6 - Discussion brings together the results of the paper and discusses the implications of research outcomes.

Chapter 2

Background and Related Work

2.1 Background

Continuous Development

Software development is a continuously evolving process. Recent decades have seen the transition from traditional development (e.g. Waterfall [Royce, 1987]), to Lean software development [Poppendieck and Poppendieck, 2003], which in turn lead to the famous Snowbird Report which presented the Agile Manifesto [Beck et al., 2001] leading to the rise of DevOps [Fitzgerald and Stol, 2015], and finally culminating in the era of Continuous Development (e.g. [Olsson et al., 2012] [Chen, 2017] [Chen, 2015]). Continuous Development incorporates and expands on Agile, DevOps and Lean practices. It focuses on reducing waste and enabling fast feedback [Fitzgerald and Stol, 2015] [Chen, 2017] [Chen, 2015] [Claps et al., 2015] [Dingsøyr and Lassenius, 2016] [Leppänen et al., 2015] [Ståhl and Bosch, 2014] [Humble and Farley, 2010]. This is done by continual automation and integration of various components of the development and operations processes. Research has identified 16 Continuous Development practices [Fitzgerald and Stol, 2015], the most common being Continuous Integration, Continuous Delivery, and Continuous Deployment. First, using common Continuous Integration practices, developers merge their work into a centralized source control repository daily [Fowler, 2016]. Second, Continuous Delivery [Humble and Farley, 2010] refers to the ability of always being ready to deploy a production version of developed product. Continuous Deployment can be characterized as a mechanism that enables, the rapid release of an organization's offering to consumers, based on a measure of time or progress, which in extreme cases up to 50 times a day

[1]. Despite the widely known benefits of Continuous Development practices, different challenges menace their adoption. In this chapter, we analyze these challenges. Throughout this thesis, I do not differentiate between continuous practices, instead grouping all of them within the umbrella term, Continuous Development, unless otherwise specified. I do this because the research objective of this thesis is to create design requirements to aid in the adoption of all continuous practices. As such, the practice being described is not relevant as the work being done encompasses challenges not specific to any practice.

Some key terminology that is used throughout the thesis is: adoption, assimilation, and diffusion. Despite being used interchangeably within some sources of literature, for the purposes of this thesis they will be explicitly defined. Adoption is the process of formally incorporating an innovation into an organization's structure. Diffusion is the process by which a technology disperses throughout an organization [Rogers, 2010]. Assimilation encompasses the entire range of first becoming aware of an innovation, research and planning around said innovation, formal adoption, and full-scale deployment [Fichman and Carroll, 2000].

Continuous Development Tools

There exist a magnitude of tools and services offered to organizations utilizing continuous development practices. This research project specifically worked with IBM's Continuous Delivery Pipeline service[2], and by extension, the Toolchain service[3]. The Toolchain acts as a centralized repository for continuous development tools including source control management, analytic tools, build and rendering tools, monitoring services etc. These are grouped into seven categories: manage, run, learn, discover, envision, build, and culture. Some of the tools and services offered here are IBM owned; however, Toolchain does allow external tool integration. The IBM pipeline follows a similar construct as most popular pipelines. A series of stages can be created and customized to build, test, and deploy a program. It allows input from multiple sources including source repositories and container images. The primary users for both of these services are IBM employees. As this research project is so closely linked with industry, this section will provide a related work for organizations producing

[1]Fitz, T., 2009. Continuous deployment at IMVU: doing the impossible fifty times a day. http://timothyfitz.com/2009/02/10/continuous-deployment-at-imvu-doing- the-impossible-fifty-times-a-day/ (accessed on January 28,2019).

[2]https://www.ibm.com/garage/method/practices/deliver/tool_delivery_pipeline/

[3]https://www.ibm.com/cloud/garage/toolchains

tools similar IBM's pipeline.

One of the most prominent continuous development tool provider is Jenkins[4]. As one of the first tools of its kind, Jenkins has established itself as the leading platform for continuous development pipelines[5]. There are many advantages to using Jenkins, it is easily the most extensible tool available. Being open source and free to use, a staggering amount of Jenkins plug-ins have been developed[6]. However, there also exist many disadvantages: being open source, there is no direct governance for the direction of development which has lead to unstable version and poor feature development. Additionally, many users claim Jenkins to have a very steep learning curve, and organizations have to devote a significant amount of resources to maintain a stable running Jenkins server [7]. Having this steep learning curve can also lead to a case of a gatekeeper, where only one person in an organization is knowledged in the complete development process.

Many large tech organizations have created their own continuous development tools as well. Microsft created Azure Pipelines[8], Amazon came out with AWS CodePipeline[9], Google added a continuous delivery pipeline to their API set [10], and Gitlab sprung up with a complete continuous development service suite [11]. Each of these services share similar features. They follow freemium pricing model– offering free a free version with the ability to pay to access additional features. Each pipeline has components for building, testing, deploying, and monitoring a product.

2.2 Related Work

Adopting continuous development is challenging because little is known about the process. Researchers note a limited scope of case studies conducted on the adoption of continuous development in the industry. In 2015 Claps et al (2015) reported the scarcity of academic literature addressing the challenges organizations face when adopting continuous development. Another impactful paper released in 2015 by Leppänen et al (2015) observe how few empirical studies exist on continuous development,

[4]https://jenkins.io/
[5]https://technologyconversations.com/2016/01/14/the-short-history-of-cicd-tools/
[6]https://ezeelive.com/jenkins-pros-cons/
[7]https://nevercode.io/blog/the-maintenance-side-of-jenkins-ci/
[8]https://azure.microsoft.com/en-ca/services/devops/pipelines/
[9]https://aws.amazon.com/codepipeline/
[10]https://cloud.google.com/solutions/continuous-delivery/
[11]https://about.gitlab.com/

and those that do focus on evaluating progress made by an organization in their assimilation of continuous practices, rather than focusing on the struggles they face. Chen (2015,2017) released a series of academic papers detailing how his organization struggled to adopt CD, citing the source for their frustrations and his work as the lack of effective guidelines for the adoption and use of continuous development tools. Within my own research, limitation easily available empirical data has been a hindrance in identifying generalizes solutions the adoption challenges. While the data that does exist comes from a wide range of organizations—varying in size, product, location, field—it does not exist in enough quantity to conduct quantitative research analysis, as solutions that exist for one set of criteria, may not be possible for another. This limitation impedes the creation of a clear set of best practices for the adoption and use of continuous development. Without these set of best practices and knowledge of challenges that organizations may face during adoption increases the barrier to adoption for many organizations. The remainder of this chapter addresses some of the challenges identified in literature.

2.2.1 Lack of Stakeholder Conciliation Hampers Adoption in Organizations

A predominant barrier to adoption is obtaining long term support from various stakeholders relevant to an organization's software development process [Neely and Stolt, 2013]. Failure to obtain support from relevant parties' dooms adoption to failure as Continuous Development is a holistic development process [Chen, 2017] [Fitzgerald and Stol, 2015]. Stakeholders who do not support continuous endeavor create a bottleneck in the software development process, introducing manual steps. Due to the holistic nature of continuous development, the existence of any manual steps within the development process creates a bottleneck [Leppänen et al., 2015]. These bottlenecks can prevent organizations from obtaining their continuous goals. For example, a case study conducted at OANDA [Savor et al., 2016] observed a 23% drop in average number of deployments after transitioning to a management team that did not support Continuous Development.

From these observations, two claims can be made. Primarily there exists a wide range of stakeholders in software development organizations, each with their own objectives and priorities [Savor et al., 2016] [Dingsøyr and Lassenius, 2016] [Fitzgerald and Stol, 2015]. Presenting the benefit of adopting Continuous Development as a

singular perspective fails to appeal to all stakeholders and diminishes chances for obtaining conciliation. A solution to this has been recently published suggesting identifying key problems with each of the stakeholder's pain-points, and make clear to them how adopting Continuous Development aids in alleviating each stakeholders pain-points [Chen, 2017]. Outside of the paper suggesting this hypothesis, I was unable to find supporting data of its efficacy. Future research could focus on the creation of guidelines to improve stakeholder buy-in.

The secondary claim is stakeholder resistance to change [Leppänen et al., 2015]. Research shows this to be increasingly prevalent for people with limited exposure to new technologies throughout their life, such as older peoples [Fitzgerald and Stol, 2015]. While developing methods to overcome this resistance and obtain stakeholder conciliation is an adoption challenge identified in this thesis, addressing cultural resistance to change falls outside the purview of this paper.

2.2.2 Continuous Development Assimilation Methods

Despite the concepts and foundations of Continuous Development being well documented, assimilation methods are not [Humble and Farley, 2010]. Moreover, there exist several Continuous Development assimilation methods, each with numerous versions, each subject to slight variations. This results in potential adopters being faced with a myriad of assimilation methodologies and no clean guidelines for which is best for them. This section provides an overview of the most prominent methods, specifically, Low Hanging Fruit, Value Driven Assimilation, and Pilot Team.

Low Hanging Fruit

Low Hanging Fruit (LHF) was introduced by Stober and Hansmann (2010). It is also known as *crawl-walk-run method*. LHF proposes teams to adopt the simplest aspects of continuous development first, and then work toward more challenging processes using their existing progress as motivation to continue evolving. This method was introduced as a theory by Stober and Hansmann (2010), and as such had not been studied in industry nor undergone the rigor of peer review at the time of publishing. However, it has been referenced by other work [Olsson et al., 2012] and described similarly in recent case studies [Laukkanen et al., 2017]. The main benefit of this method is diminishing overhead organizations face when adopting Continuous Development. In an effort to achieve this, teams evolve minor aspects of their existing

process, through techniques like automation. This process can be understood as serial incremental adoption, as opposed to parallel adoption. The appeal of this method to management is this low upfront cost, which is especially true for those on the fence about adoption.

Value Driven Assimilation

The next method focuses on the value an organization can obtain from adopting the various aspects of continuous development [Eck et al., 2014] [Svensson and Host, 2005] [Conboy and Wang, 2007]. Within the context of this study, it will be referred to as the Value Driven Assimilation (VDA) method. An observation made via multiple case studies of adoption [Eck et al., 2014][Svensson and Host, 2005][Conboy and Wang, 2007], found the order in which organizations adopted Continuous Development practices was based on the perceived value gained. This value was found from an amalgamation of customer value, organization value, and value to the development team. While similar to LHF, as both are benefit oriented, VDA seeks to optimize value gained as a priority, not ease of adoption. It is critical to note that while these studies show similarity in efficacy, the results obtained are subjective, based on interviews with developers, managers and testers (ex. [Olsson et al., 2012] [Leppänen et al., 2015] [Svensson and Host, 2005] [Savor et al., 2016]). Hence the use of the term perceived benefit, the importance, impact, and benefit of these practices being adopted are primarily determined from opinions of those interviewed with little empirical data presented.

Pilot Team

The final assimilation method covered by this thesis is the use of a Pilot Team. This method is best exemplified by the seminal papers written by Chen (2017,2015). In these papers, Chen speaks of his experience as a developer within a large organization, Paddy Power. Chen was placed in charge of orchestrating the adoption of Continuous Development for Paddy Power. He discusses the challenges Paddy Power faced, and the methods used to overcome them. While the use of a Pilot Team for experimenting with emergent technologies and methods is not new, Chen's paper (2017) details a multi-year, first person experience, giving a unique perspective into this situation. Additionally, Chen also presents abundant empirical data regarding benefits and costs of multiple aspects of the transition to continuous development specific to Paddy

Power. The Pilot Team method is where an organization creates a team of developers with a breadth and depth of organizational and technical, knowledge and experience. The Pilot Team selects a less complex offering, and undergoes the assimilation process for Continuous Development. One company studied proclaims that the use of a pilot team serves as guiding light, an inspiration, for the rest of the organization "Having a positive effect on other teams." [Olsson et al., 2012]. Chen (2015) noted that some challenges faced are unique to specific organizations and having a Pilot Team undergo the adoption process creates a knowledge repository for future teams to engage in a knowledge exchange reducing the overhead faced by those teams.

While each of these methods has separate and distinct processes and objectives, there exists variations and combinations specific to each case studied. As an instance of this, Chen (2017) proposes using such a combination in his work. He describes using a pilot team to start the assimilation process, with that team focusing first on tasks with the most efficient cost/benefit ratio. However, by incorporating all of these methods organizations may incur each of the negative effect as well: loss of value from choosing a process to adopt that does not bring the largest value first; delayed value from increased initial overhead; and delayed organization-wide benefit due to time taken by pilot team instead of all teams adopting in parallel. In order to make a determination about any of the methods proposed there exists a need for empirical data of the cost and benefit for each of the methods of adoption. Stahl and Bosch (2014) created a model to provide this information. They conducted a broad survey of industry practices and created a model to track change within an organization before, during, and after the adoption process. Future research could apply this model to a large enough set of organizations, creating a generalizable, empirical set of cost/value equations for each of the methods of adoption. Organizations could use this empirical-based model, and apply a weight specific to their non-functional requirements, yielding an informed set of outputs: the method of adoption best suited to their needs, expected cost for that method, expected value for that method.

2.2.3 Cultural Resistance To Change

Cultural resistance to change presents substantial challenges for the adoption of continuous development within an organization. Cultural conventions are often deeply rooted and can seem immutable. There is an abundance of references made within literature to the problems associated with cultural resistance; however, this section

will categorically discuss 3 of them and their direct association with continuous development adoption. Stolt and Neely (2013) case study documenting their adoption process for continuous development make multiple references to cultural resistance encountered. From overcoming the initial question of "why bother", to an intern's reaction of "Deploy to production? Automatically? That's scary!", the diversity of cultural resistance is intrinsically challenging.

The two prominent cultural challenges identified by the study of related work in the field of continuous development that will be discussed in this section are as follows:

1) Lack of trust in the development system as well as lack of confidence in its ability to catch potential issues.

2) A pervasive culture of shame and blame in the software development community.

Lack of Confidence and Trust

A lack of confidence in the software development infrastructure's, typically testing, ability to prevent errors from reaching production creates an environment of mistrust dissuading developers from ever attempting to reach a fully automated state. There is ample evidence of this lack of confidence evident in literature studying cases of adoption. Neely and Stolt (2013) observed that before an organization could achieve continuous developers must first place trust in automated tests. It was noted poorly implemented automated tests, with low coverage, diminishes trust [Leppänen et al., 2015] leading to the implementation of manual steps in the development process. Trust is further diminished through the existence of "flakey tests" – non-deterministic tests, or tests which fail inconsistently [Neely and Stolt, 2013]. "Flakey tests consist of failures related to "timing issues, transient problems such as network outages, test/code interaction, test environment issues, UI tests or determinism or concurrency bugs. Flakey tests have caused a lack of discipline and ambiguity in test results."[Laukkanen et al., 2017]. Chen (2017) noted the existence of these flakey tests often resulted in additional levels of manual testing to verify the results of the content the tests were suppose to cover. All of these contribute to a diminished level of confidence in the development infrastructure, creating a mindset of mistrust.

As indicated the harm in having said mindset is a diminished capacity to increase development velocity, halting assimilation of continuous development. An ideal situation would be to convince everyone to trust the system; however, that is not feasible.

Instead, continuous development tools could be expanded to provide features that address this lack of trust. This includes providing developers an alternative means of verification instead of resorting to manual steps being added to the development process.

Culture of Blame and Shame

While not as predominant as the other themes, careful study of related work in the field of continuous development uncovered multiple indications that there exists a culture of blaming and shaming those who create user impacting problems. "Also, developers' reputations are on the line: deploying a broken build to customers could strain the relationship between parties and create an unwanted user experience. Any lack of confidence in an application's quality is amplified by the knowledge that any and all changes are immediately deployed."[Leppänen et al., 2015]. When this mentality exists within an organization it impedes the speed of innovation, contrary to continuous development's primary objective. In both Stälh and Bosch (2014) and Laukkanen et al (2017) systematic literature reviews, *fault handling* is identified as a theme within the literature; however, it is discussed as technical implementation-ie which non-technical solutions are used when handling faults. Ståhl and Bosch (2014) further groups *fault responsibility* as various means of identifying who is responsible for dealing with a fault. An example of such would be; author of latest code check-in. Having guidelines within an organization for both fault handling and fault responsibility are effective means to minimize damage incurred from faults; however, lack a means to prevent future faults from occurring.

While the negative impact of this mentality is not easily identified, the positive impact of doing the reverse is. A case study at Facebook and OANDA revealed what they call a culture of *no-blame* [Savor et al., 2016]. Faults detected in the development process undergo a no-blame postmortem where developers are encouraged to identify which step in the process allowed the fault to occur, then focus on improving the automated test suite in order to prevent further faults from occurring. Both organizations, Facebook and OANDA report this empowers developers to move fast, spawning innovation at a much greater scale [Savor et al., 2016].

Currently missing is a system to enable this within the software development toolkit. The current tools allow for traceability of a problem—helping to identify who was the source of the problem—which is important for many reasons. However,

this enables the culture of blaming an individual and attempting to solve them, rather than focusing on solving the process which allowed such a mistake.

2.2.4 Related Work Using Design Science

While Hevner's (2007, 2004) method was created an intended for research being conducted in the field of information systems, this research is not the first to adapt his work for use in software engineering. Roel Wieringa created a refined framework of the 3CV defined in context of software engineering [Wieringa, 2009]. More recently, this same design science methodology was implemented in another joint IBM project [Montgomery, 2017]. In Montgomery's project, Hevner's three cycle iterative design science methodology was used to characterize a challenge in industry (support ticket escalation in IBM's ecosystem), learn from the application domain (IBM customer support experts), design a solution and validate an artifact (escalation prediction application).

Chapter 3

Research Methodology

To address the research questions presented in this thesis, design science principles first introduced by Henver et al (2004) for information systems were used. Henver advanced his design science principles to include the Three Cycle View of Design Science Research (3CV) [Hevner, 2007]. 3CV is an iterative model that uses three cycles to create an artifact built on the understanding of both academic and domain specific knowledge. The created artifact is then iteratively evaluated and verified to increase quality and maintain relevance and direction to the research objective.

Using 3CV, the research had 4 distinct components: Learning - Application Domain, Learning - Academic Literature, Artifact Creation, and Artifact Evaluation. Learning from the application domain utilized industry experts, such as IBM developers, to guide and validate the scope of research. Incorporating industry experience allows the research to maintain relevancy to evolving stakeholders' needs. This is what is referred to as the Problem Characterization phase of this research. This process of incorporating information from Problem Characterization is the *relevance cycle*. A extensive study of literature began once the scope of the research was determined. Continuous investigation of existing literature within the research scope contributed to the knowledge base; aiding the advancement of the research objective. This process is the *rigor cycle*. Artifact creation and evaluation draws resources from both *relevance cycle* and *rigor cycle*. Artifacts—UI implementation, surveys—are created to address the research objective identified in the *rigor cycle* and *relevancy cycle*. Knowledge from existing research informs the artifact creation through the rigor cycle. Similarly, artifact evaluation relies on both the *rigor* and *relevance cycles* to ensure the artifacts satisfy both the research objective and requirements of stakeholders. The integrated nature of artifact creation and artifact evaluation is

called the *design cycle*.

3CV was chosen because it reflects the values of the research objective. Continuous development's primary objective is the rapid delivery of value, allowing for feedback to guide direction. 3CV's iterative approach allows for small batch, incremental progress to be guided by early feedback. This has enabled easy pivoting as the scope of the project has evolved. It also provides first-hand insight into the value of obtaining early feedback. In the software industry, this is referred to as "eating your own dog food" or "dogfooding"- simply put, when one uses the artifact they create.

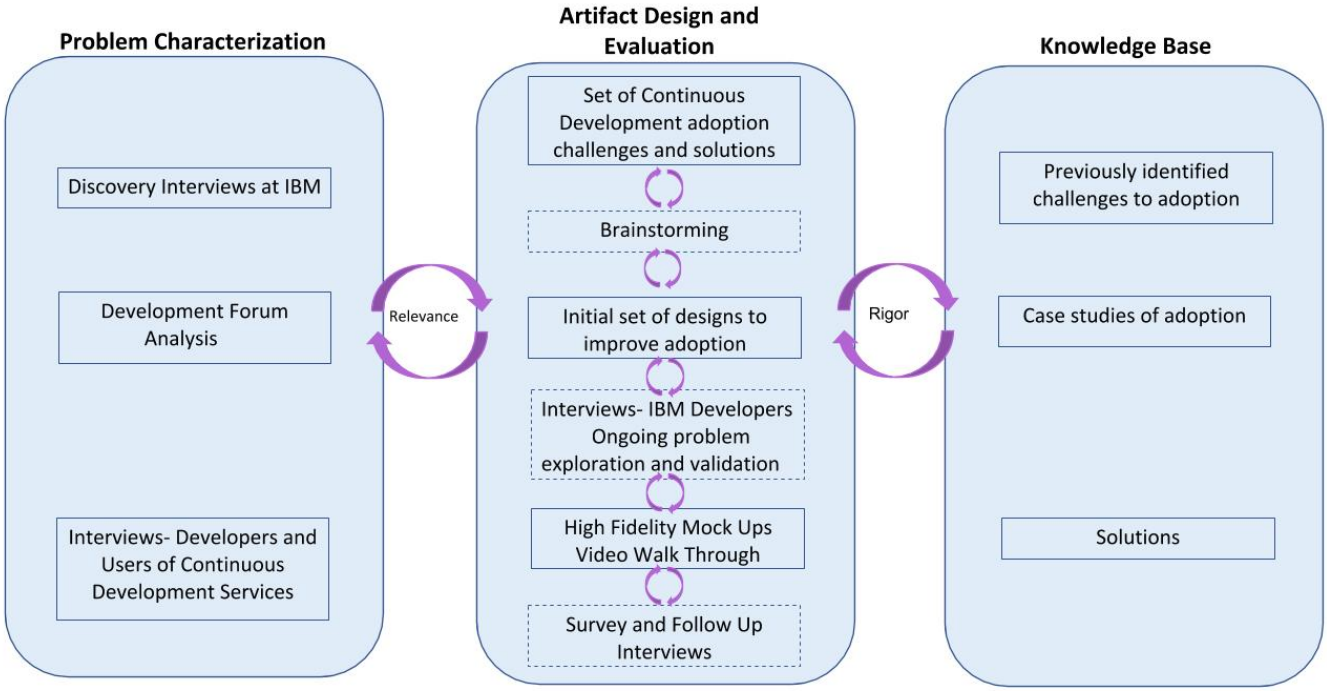

Figure 3.1: 3CV Research Methodology

3.1 Problem Characterization

The research began with multiple relevancy cycles. These cycles helped to characterize the challenges associated with Continuous Development adoption. The wide range of challenges and solutions help inform artifact development. Exploratory interviews focused on problems with the IBM Continuous Delivery Pipeline from the perspective of its developers. This generated an initial set of adoption challenges. Next, I analyzed

developer forums to expand on that set of challenges. Utilizing a source external to IBM provided insight into challenges that were not specific to IBM's pipeline. These steps generated what I considered to be a complete list of adoption challenges. At this point, the set of adoption challenges was refined based on technical feasibility to implement, time/resource constraints, and potential impact on adoption. The follow-up interviews conducted at IBM was the fine step in selecting the challenges this thesis addresses. These challenges are described in Section 4.4 The Problem.

3.1.1 Exploratory Interviews

This research project began with a relevance cycle. I met with several IBM developers to explore areas where research could be applied to improve IBM's Continuous Delivery Pipeline and help IBM developers, users of IBM products, and stakeholders in the continuous software development domain. These meetings happened at IBM development offices. The interviews were conducted with 7 members of the IBM Continuous Delivery development team whose positions included: Chief Architect(1), Senior Software Developers(2), Junior Software Developers(3), Product Owner(1). Over the course of these interviews an extensive preliminary set of adoption challenges, and solutions were developed. The full process and results are located in Chapter 4.1.

3.1.2 Developer Forums

The second relevance cycle studied popular software engineering forums. While an initial list of adoption challenges and solutions existed from the exploratory interviews, I wished to expand the scope of challenges considered to those provided by the larger Continuous Development community. Stack Overflow[1] was selected based on stakeholder recommendation and significancein the software development community. The following search terms were used:

"Continuous-Deployment"

"Continuous-Integration"

The resulting posts were then sorted by view count — the amount of traffic each of the posts had accrued. This was done to prioritize issues of higher significance. 612 posts were analyzed and sorted into 6 categories The complete process is discussed in Chapter 4.2.

[1] https://stackoverflow.com/

3.1.3 Interviews

After the initial iterations of artifact creation and evaluated were completed, a series of follow-up interviews were conducted at CASCON 2018, a computer science conference hosted by IBM. These interviews served two purposes: a means of artifact validation and to collect information contributing to problem definition. Interview length was originally 1 hour; however, several follow-up interviews were required. Interviews were conducted at IBM development offices in Toronto, Ontario, Canada. Interviewees provided feedback on the various design concepts and implementations. This feedback included technical feasibility, demand/interest in a given solution/problem, and alternative approaches. Additionally, these discussions revealed new challenges. The result of these interviews helped to shape the artifacts' development direction (Chapter 4.3).

These interviews were recorded with consent and transcribed anonymously. To anonymize the transcripts, interviewee's names and identifying details such as role/position were replaced with a key from the transcripts. A reference key matching those details and names to the interview was created and stored within an isolated University of Victoria, Segal Labs server.

3.2 Artifact Development and Evaluation

3.2.1 Iterative Problem Development and Evaluation

Since project conception, semi-weekly meetings with a senior industry partner, Mike Wilson, have occurred. The objective and scope of these meetings have changed over the course of the research. Initially, these meetings were used to guide the research towards valuable resources such as industry experts, interview candidates, and relevant literature. Later, they became a continuous means of gaining valuable feedback for the initial set of design requirements to overcome adoption challenges.

3.2.2 Survey

A survey was created with 3 intended outcomes:

1) Establishment of current industry practices
2) Evaluation of adoption solutions' theorized benefits
3) Evaluation of design requirement implementations for adoption solutions

The survey was distributed across multiple mediums sequentially. Initial distribution was to a group of internal IBM employees, most of whom were involved in previous interviews (see subsection 3.1.3). This group provided responses to the survey and feedback on question structure and formatting. The first external platform was the /r/devops reddit subthread [2]. This was used as a means to test reactions from the software development community. The survey was then distributed to software development facebook groups: "Hackathon Hackers" [3] and "Cirque du Twerque" [4]. The final survey destination was reddit subthread /r/cicd [5], suggested by the reddit community as a source of additional survey participants.

The survey was developed and published using Google Forms [6]. Google Forms was chosen for its multiple methods of survey distribution and the ability to perform simple analysis of survey results, allowing for rapid initial feedback and insight. The survey was created and distributed in compliance with the University of Victoria Ethics and Consent agreement.

Establishment of Current Industry Practices

Sections 1, 2, and 3 of the survey were designed to obtain empirical data on the users of continuous development tools. These questions were aimed at discovering the following:

Development processes and practices

Tools and services used in industry

Efficacy of continuous methodology in development teams

Evaluation of Adoption Solutions' Theorized Benefits

Respondents were presented with a set of design requirements to assist in overcoming adoption challenges. Each design requirement presented was accompanied by questions pertaining to its usability, benefit, and disadvantages. Space was given for users to provide any additional comments on the design requirements.

[2] https://www.reddit.com/r/devops/
[3] https://www.facebook.com/groups/hackathonhackers/
[4] https://www.facebook.com/groups/HHCirqueDuTwerque/
[5] https://www.reddit.com/r/cicd/
[6] https://docs.google.com/forms/

Evaluation of Design Requirement Implementations of Adoption Solutions

For each of the design requirements, a high fidelity design of a possible implementation was created. A video tutorial walk-through depicting the use, functionality, and possible use cases of the set of design requirements were created. This allowed for very specific feedback to be provided on the design requirement's validity and the specific implementation of the design requirement within IBM's pipeline.

3.2.3 Survey and Follow Up Interviews

A final series of interviews was conducted to determine the viability and usefulness of each design requirement. Final versions of the design requirements created, accompanied by their syntactical logic, were presented and discussed. Follow up interviews were conducted at IBMs development offices in Markham Ontario and well as remotly with several developers not working wit hIBM. Interviewees were shown each of the design requirement video walkthrough. Next a question and answer period as they inquired about each design. Finally, they were asked if the designs would add value to their development practice, if it would enable them to remove manual processes, and if the specific UI met their needs and requirements.

Chapter 4

Problem Characterization

Before a solution to a problem can be found, the problem must be clearly defined and scoped. This chapter describes the process of defining, characterizing, exploring, and validating problems faced in continuous development adoption, as stated by the second research question (RQ2).

4.1 Exploratory Interviews

I traveled to Ottawa, Ontario, Canada in June of 2018 to meet with members of the IBM Continuous Delivery team, which consisted of 6 full-time software developers and a team lead. These 7 developers were each responsible for specific areas of product development. Due to availability constraints, I spent the majority of my time working with Senior Technical Staff Manager (STSM), Mike Wilson. He provided a basic overview of the IBM Continuous Delivery offering. IBM Continuous Delivery is a sub-team of the larger Toolchain service. Toolchain offers the complete DevOps package, including source control management, issue tracking, code integration, test suites, integrated cloud IDE, pipeline, monitoring, and performance evaluation metrics. Toolchain is a growing platform where developers can obtain any services they might need to design, write, test, deploy, and monitor their product. The team with which I collaborated was primarily focused on the pipeline service. The IBM Continuous Delivery Pipeline [1] is a freemium (free for basic use with paid premium features) service which allows any user application—from an individual's website to an enterprise's warehouse stock tracking applications—to be integrated from a source repository,

[1] https://www.ibm.com/garage/method/practices/deliver/tool_delivery_pipeline

run through a series of test suites, and deployed to an environment where it can be monitored. All of these actions are implementable on their own; however, use of a pipeline has several key advantages. The first advantage is the grouping of all of these steps, which enables simple and consistent modifications to be made to the application through use of features such as shared global variables, API keys, and credentials. Secondly, the use of a pipeline allows for rapid deployment of changes. The structure and flow of a pipeline enables users to commit changes to their source repositories and have that change permeate to the end-stage environment with no additional manual interaction. This enables high-performing organizations to commit changes at a reported rate of 50 times per day [Andi Mann, 2016].

After obtaining knowledge of how the pipeline worked within the Toolchain, I attending meetings with 4 other full-time developers as well as a service offering analyst. This process provided insight into team dynamics, current processes, and problem areas. As previously mentioned, this team was distinctive in their use of dogfooding—the process of using the tool they are creating. This gave each of the developers unique insight into the advantages and disadvantages of the IBM pipeline tool, as they have encountered the difficulty with the lack of functionality, but also are aware of can be implemented feasibly. This insight, combined with user feedback obtained via the service offering analyst, allowed me to create a "wish-list" of pipeline functionalities not currently available that would improve the overall adoption. This list is displayed in Table 4.1 and became the first artifact developed in the Problem Characterization step of this thesis.

Pipeline-As-Code	A means of interacting with the pipeline though a script file, avoiding use of the UI entirely
Embedded IDE	An Integrated Development Environment (IDE) within the pipeline UI, allowing modification to source code
CLI with CRUDE Operations	A Command Line Interface (CLI) allowing for create read update delete and execute (CRUDE) operations on the pipeline
Toolchain Level Environmental Properties Tile	Extracting environmental variables from the pipeline stages to the Toolchain- increased security and ease of use
Parallel Stage Execution	The ability for sequential stages within the pipelines to run in parallel - decreasing pipeline run time
Traffic Distribution Management Tool Integration (Ex. ISTIO)	A tool that allows an organization to dictate how much traffic goes to each of its running applications
XCode Integration	Enables developers to code and deploy software developed in XCode
Developer Versus Administrator Pipelines	A separate view of pipelines, developer with technical minutia, administrator as an overall health of the system view
Additional requests were to enhance existing features:	
Advanced Logging	An easier to use and more enriched logging interface
Permission Control extended from Pipeline to Stage Level	Allow for permission control to be implemented on specific stage within a given pipeline

Table 4.1: Continuous Development Adoption Challenges. This table explores the adoption challenges identified during exploratory interviews.

Each of the issues presented in the table above was then analyzed to determine research viability. Criteria considered were: impact on adopters, technical feasibility, research applicability, relevancy, and prevalence. Pipeline-as-code and XCode integration were disregarded, as these features were already set to be addressed by IBM in the near future.

The next steps would be validation and exploration of issues in the continuous development user base. To this effect, it was determined that additional information would be collected from the continuous development user base through an analysis of developer forums. The intended outcome would be a prioritized list of continuous development adoption challenges.

4.2 Analysis of Developer Forums

Collecting data from developer forums enabled identification, validation, and improved characterization of issues described in Section 4.1. Additionally, I ascertained the existence of current continuous development adoption solutions.

Stack Overflow was selected as the source of developer forums because of its prevalence in the software development community. Diversity in the populace of Stack Overflow users indicated a robust sampling, allowing insight from additional stakeholders in the software development life-cycle. Figure 4.1 shows the variance in roles for users of Stack Overflow's services [2]. The data shown in Figure 4.1 was collected through a survey conducted by Stack Overflow in 2019. Stack Overflow offers services to optimize data collection, including key word searching, sorting by criteria such as most recent post, view count, up votes (a counter representing the number of users who found the post valuable), and the number of responses. This search optimization enabled prioritization of data points gathered to be those most relevant to the research objective.

Utilizing the aforementioned services, a search was created to parse the Stack Overflow database using the following tags:

[continuous-deployment],[continuous-integration] [3].

This search returned 619 forum posts. Posts were sorted. Criterion for sorting posts was the number of **up-votes** each post had. Analysis was halted after 322 posts due to diminishing return. After an initial manual analysis of the forum posts,

[2]https://insights.stackoverflow.com/survey/2019

[3]https://stackoverflow.com/questions/tagged/continuous-deployment+continuous-integration

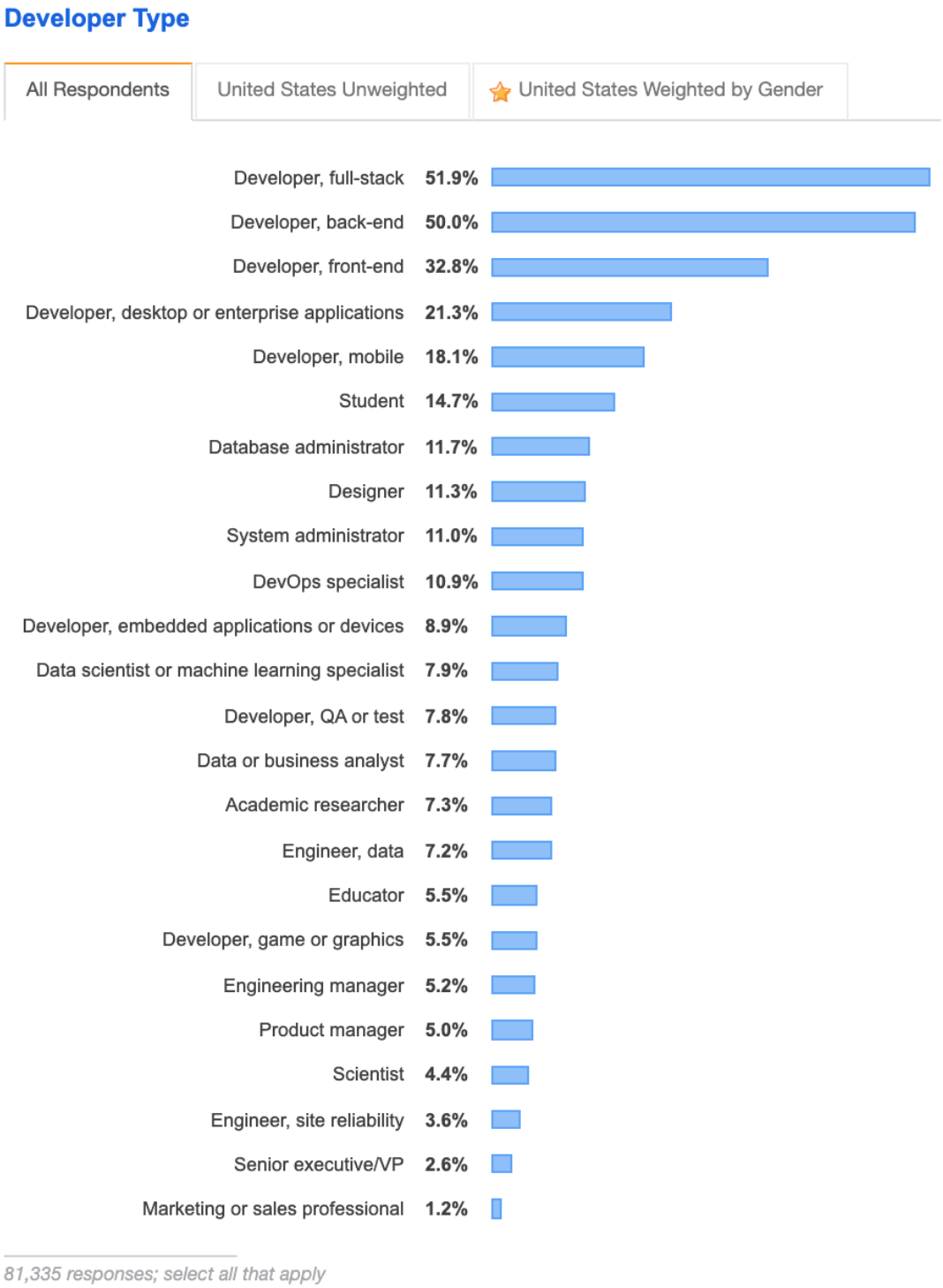

Figure 4.1: Stack Overflow Developer Survey 2019- Developer Type Overall

categories were created and each of the posts was sorted into a category (Table 4.2).

Category	Summary
Feature Requests	Posts seeking specific functionality within a given continuous development service/tool.
Set Up Guidance	Adoption questions presented from users intending to, or in the process of, adopting a new technology. Often advice-based questions seeking guidance from users experienced with their adopted technology.
Clarification Of Terminology	Users presented questions to the community to clarify the specific definition, process, and meaning for Continuous Development specific terms.
Best Practices	Advice-based posts either asking for, or offering, best practices for developing software with a given set of parameters.
Technical Questions and Error Resolution	Users asked questions about specific technical problems they encountered (error codes, how to do something).
Non Relevant	Forum posts that were not related to Continuous Development- discarded.

Table 4.2: Developer Forum Post Categorization

Feature Requests

Reduced pipeline build time was a common request. This request may seem senseless as build time varies on the size of the project being built. However, it inspired reflection on methods to improve flow through a pipeline. An example of this improved flow is parallel stage execution—a challenge identified during exploratory interviews.

Also requested were additional methods to instigate pipelines. Two examples of this request were: triggering builds off special characters in git commit messages[4] and using git triggers[5]. Git tags are unique identifiers for a Github repository that

[4]https://stackoverflow.com/questions/5281816/trigger-build-in-jenkins-hudson-using-hashtag-in-commit-message

[5]https://stackoverflow.com/questions/44003948/jenkins-how-to-trigger-pipeline-on-git-tag

allows actions to be made if a tag is included for a commit. For example, if I had the tag "Production" and included that in my git commit my pipeline could theoretically read that and direct that specific build version to the production environment. These forum posts indicated the need for advanced control over pipelines' end-to-end flow. This need led to deeper questioning of what it means to control pipeline flow and what level of flexibility could be implemented with current pipeline technology. Some desired features—identified during exploratory interviews—addressing this issue included Traffic Distribution Management Tool Integration, CRUDE pipeline operations, Embedded Integrated Development Environment, and Permission Control extended from pipeline to stage level.

Best Practices

Best practices forum posts were particularly interesting, as best practices for Continuous Development is a core component of this research. They consisted of giving or requesting advice on methods to improve continuous development processes. Often, the posts were narrow in scope, such as tagging convention for docker images in continuous deployment projects[6] or deploy rate using high bandwidth images [7]. Others addressed broad topics like best practice for dealing with multi-branching, merge/test/deploy frequency, and syncing multiple code bases per deploy[8].

Reviewing best practices forum posts provided evidence supporting an adoption challenge discussed in literature—there is no clear simple assimilation path for continuous development adoption [Humble and Farley, 2010]. Those attempting to adopt continuous development struggle to find solutions to challenges they will face [Chen, 2017]. Instead, adopters utilize online forums to post their questions and review anecdotal advice [Claps et al., 2015].

Technical Questions and Error Resolution

While most technical questions and error resolution forum posts were case specific and did not contribute to the research objective, one theme did emerge: deprecation of services and features affecting usability of an application for a portion of the user base.

[6] https://stackoverflow.com/questions/33196382/what-is-the-best-docker-tagging-strategy

[7] https://stackoverflow.com/questions/27642412/docker-build-push-every-time-is-it-practical-for-continuous-deployment

[8] https://stackoverflow.com/questions/9105459/best-practice-for-continuous-integration-and-deployment

Organizations are discontinuing features and components of their services/tools/offerings that are still in use/demand by a portion of the software development community. This may be indicative that either industry isn't listening to users' needs, or that users have an unrealistic expectation of product support.

Set Up Guidance

Set up guidance posts did not provide relevant adoption challenges. However, these posts outlined a potential topic for future research. A common theme in set up guidance questions was inexperienced users seeking help/advice from experienced users for a given tool/service/process/offering. The existence of this theme is significant. It demonstrates that inexperienced users are turning to community driven support groups (Stack Overflow) for advice. Future research could determine how this came to be, and the viability and continuity of inexperienced users relying on community forums as a primary source of adoption guidance, rather than evidence-based best practices.

Clarification of Terminology

Clarification of terminology posts generated a staggering amount of votes and views. They revolve around a central theme—a lack of clear definitions for terminology used in Continuous Development. The most viewed and voted forum post was asking for clarification on the difference between Continuous Delivery, Continuous Integration, and Continuous Deployment. At the time this data was collected, the post had 89,000 views on it (Sept 2018), now 124,000 (Sept 2019). This lack of definition for continuous development terminology and processes is a sentiment echoed throughout exploratory interviews with IBM engineers. Many sources use Continuous Integration or CD as an umbrella term to define all continuous practices. This misleads users, implying that each term can be used interchangeably, which is not the case.

4.3 Interviews

A series of follow-up interviews were conducted at IBM's Center for Advanced Studies Conference (CASCON) in 2018. These interviews served two purposes: a means of validating problems identified thus far, and collecting information contributing to further problem definition. Interview length was one hour with several one hour

follow-up interviews required. Interviews were conducted at IBM development offices in Toronto, Ontario, Canada. Interviewees included members of a different team operating within the IBM Toolchain Platform, as well as several IBM developers who worked with IBM Continuous Delivery and expressed interest in the research being conducted. Interviewees provided feedback on day-to-day life as software developers adopting continuous development practices and the specific challenges they encounter. These discussions revealed new challenges to be explored.

These interviews were recorded with consent and transcribed anonymously. To anonymize the transcripts, interviewees names, and identifying details such as role/position were replaced with a key from the transcripts. A reference key matching those details and names to the interview was created and stored within an isolated University of Victoria, Segal Labs server.

These interviews began with a standard series of questions—name, title, years of experience in that role, years of experience at IBM, and years of experience in software engineering. Following this, inquiries were made into the employee's experience with, and opinions of, continuous development.

4.3.1 Interview Responses

Define continuous development/Do you practice continuous development

Three of the six interviewees were able to describe a continuous development practice (Continuous Integration, Continuous Deployment, or Continuous Delivery). One of those three was able to accurately describe the procedural steps of the term they provided, according to definitions previously defined in the manuscript by Humble and Farley (2010). The three that could not describe a continuous development practice described their own development process, which, by accepted definitions, were not continuous practices. This lack of understanding of the terminology and processes associated with continuous development is consistent with what was found in developer forum analysis. Two of the respondents believed they were continuous, two defined their process as partially continuous, and the final two said they were not practicing continuous development.

Describe a typical deployment experience

This question presented enormous discrepancy in responses. Despite each interviewee being part of the same development team, each had unique continuous development experience. One interviewee continuously developed, tested, and integrated software independently. Another was assigned tasks, developed, submitted a pull request, and had no further involvement. Yet another interviewee hardly developed at all; however, he took responsibility for managing their team's manual deployment to production.

Do manual steps belong in continuous development processes

Four of the six interviewees said there should be no manual steps for continuous development, one stating, "At the end of the day, if there is any one thing that is crucial to the continuous delivery of a change or a set of changes that isn't itself able to be done fairly continuously then your screwed, you've got a kink in the armor, a weak link in the chain". However, when asked why manual steps existed in their development processes, each of the four interviewees provided very interesting insight as to why. Section 4.3.2 provides an analysis of the responses given.

4.3.2 Interview Discussion

The following questions were not scripted. The intent of these questions was to dive deeper into what was holding the interviewees back from fully adopting continuous development and explore potential solutions to enable full adoption.

We began by asking why the four interviewees thought manual steps existed in their development process and a dominant theme emerged: **lack of confidence**. This lack of confidence presented itself in two key aspects: technical and cultural.

Lack of Confidence - Technical

When asked if the interviewees would feel comfortable removing manual steps and pushing directly to production, all four interviewees indicated they were not confident that the system would catch all bugs, in all cases. Three of the four respondents indicated they would be confident to push minimal impact changes directly to production; however, the infrastructure to support such functionality did not exist at the time of these interviews. Another challenge—lengthy build time—caused an increase in

rollback time if an error made it to production. Two interviewees indicated they would be more open to removing manual steps if they could fix mistakes faster.

Improving Technical Confidence

We then asked each of the interviewees what, if any, technical implementations would help to increase their confidence to remove manual steps from their deployment process. Their responses were absolutely critical to the research objective as they yielded crucial answers regarding how to help organizations increase their confidence in removing manual steps and allowing them to develop more continuously. Means to improve technical confidence include:

Increase in test automation, coverage, quality

Advanced logging capabilities

Better deployment options (blue/green or canary)

Testing environments more closest match production environments

Broad scope and automated gates

Lack of Confidence - Cultural

Perhaps the most substantial barrier to continuous development is an organizational culture of risk aversion [Ries, 2011]. In all six interviews conducted, each interviewee mentioned an aversion to rapid continuous deployment without manual steps due to potential backlash for causing downtime for users. This compounds on the culture of shame/blame described in the related work chapter. Three of the interviewees, unprompted, spoke of a desire to change their organization's culture to one that focuses on a fix-forward mentality. When asked why they thought their organization was not practicing continuous development an interviewee responded as follows: "Why are we not doing continuous delivery? You're preaching to the choir. I would love to do continuous delivery. I would love to check-in and do nothing and at one point get feedback from production saying you failed or succeeded, but my feeling is the business is risk-averse, the business is worried about breaking production if everyone pushes like this." When asked why they thought that they responded: "I'm just telling you how it works and what I think the business believes and the fact that right now when something goes wrong in production the executives go 'Woah we broke that' and instead of saying you guys fix that so it doesn't happen the next time they say okay lets create some gates so we make sure it cannot happen. Making sure we all test

together and we all push together and we do all that together because we don't want that (downtime) to happen, that's money. Same problem with security and privacy."

Improving Cultural Confidence

Changing an organization's attitude towards a process is no easy task. As one interviewee put it, "You could hand us a paper or report that explicitly, definitively, says having no manual steps in our development process is 100% the correct thing to do, we would still keep doing things the way we have ... Changing tools is quick, changing people takes time." Nonetheless, we asked each of the interviewees if they had any ideas of technical solutions that would empower them to develop more continuously. Means to improve cultural confidence include:

Have requirements for changes and improvements to code be discussed and approved by all stakeholders (Including users)

Allow only specific people to complete specific tasks (Fine Grained Access Control)

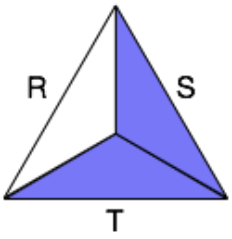

Figure 4.2: Scope and Time

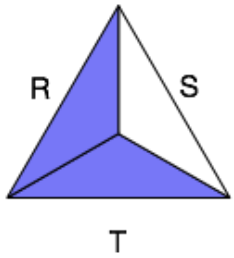

Figure 4.3: Resources and Time

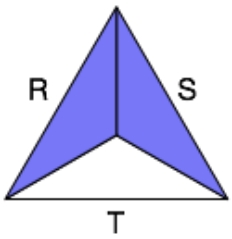

Figure 4.4: Resource and Scope

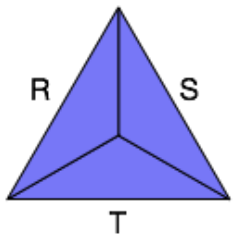

Figure 4.5: Resource, Scope, and Time

Shoe Maker's Paradox

Another barrier to removing manual tests was time and knowledge. One interviewee stated: "I wish I had the time and knowledge to automate all of this testing". When asked if knowledge or time was the limiting factor the interviewee responded "definitely time, with enough time I could learn how to do it, the overhead is what keeps me from automating my manual tests" This sentiment was echoed by 3 other interviewees directly, one going on to explain the culture of continuous development as the "Shoe Maker's Paradox" more commonly referred to as Project Management Triangle. In the project management triangle describes the constraints of project development. It consists of 4 factors following one simple principle, the quality(1) of work is constrained by the projects resources(2), deadline(3), and scope (feature)(4). The idea is it maximize quality by adjusting each of these constraints. However, changes in one constraint directly affects another constraint. In the context of software development, it is possible to directly link software development methods to specific constraints.

The quality of work is constrained by the project's budget, deadlines and scope (features). The project manager can trade between constraints. Changes in one constraint necessitate changes in others to compensate or quality will suffer.

Figure 4.2 illustrates a software development process were the scope and time are preset but resources is flexible. This is indicative of classic waterfall development, where, nearing a set deadline, organizations would pull in more resources (developers) to make sure the product was released on schedule.

Figure 4.3 is a software development process where time and resources are constrained. This best exemplifies and agile development style. Developers practicing agile have set frequencies in which they must commit/publish code. In order to make that development style work it often involved pushing commented out code, or non-functional components, ie reduced scope.

Figure 4.4 is theoretically where continuous development lives. The scope size and resources are limited; however, by developing and publishing updates incrementally, it removes the time constraint.

After interviewing all six developers, Figure 4.5 painted a clear picture of the actual culture of development where organizations constrain resources and scope but still expect deadlines to be followed in the same way they were in Waterfall and Agile development.

While the shoe maker's paradox does not directly validate any of the existing design concepts, it reinforces the importance of alleviating adoption challenges to minimize time, resources, and cost developers face when adopting continuous development.

4.4 The Problem

After extensive review of data collected and literature, four salient challenges to the adoption of IBM's Continuous Delivery Pipeline were identified. These four challenges, as detailed in Section 4.4, were formed from the problem characterization steps in this chapter. I used these challenges as the basis for the design requirement artifact. Where each design requirement is focused on solving the challenges identified.

4.4.1 Lack Of Confidence - Technical

Software development stakeholders are unassured their deployment infrastructure will prevent errors from reaching production, ensuring consequences from continuous automated deployment. The observed response has been users embedding extensive manual operations within the development process, limiting an organization's ability to develop continuously. A proposed solution is the implementation of control gates

within the pipeline. Control gates act as independent entities within the pipeline. These gates attach to stages and prevent execution until a given set of criteria are met. Control gates provide increased control over pipeline flow, enabling automation of manual processes such that if manual authorization is required, it allows the administrator to quickly review relevant data to the execution of a stage.

Reducing build time was a common request in both developer forums and exploratory interviews. One interviewee stipulated a reduction in pipeline run time would increase their confidence in deploying more continuously as their ability to deploy fixes to compromised components would increase. Optimizing a specific user's build time was far too narrow a scope for this research project, thus I began searching for a means to optimize pipeline run time for all users. I found one method to achieve this. Many developers spoke about the time-consuming nature of tests in continuous development tools. Due to the nature of test automation, most organizations opt to analyze and test as much of their code base as possible in lieu of manual testing. This results in a staggering test execution run time. One reported case found the developers pushing upwards of 20 changes a day, with their pipeline only able to test and push 5 in a 24 hour period. Upon close inspection of the pipeline's functionality, it was discovered that certain test/test stages could be executed in parallel to reduce the pipeline run time.

4.4.2 Lack of Confidence - Cultural

In order for continuous development to function as intended, it requires all stakeholders in an organization to function with a continuous development mindset. Unfortunately, this chapter uncovered a plethora of examples where this is not the case. Risk-averse stakeholders can slow down the entire organization's development speed. It is certainly a case of deploying only as fast as your most hesitant developer. The design requirement created to address this was a modular design framework for pipeline access. For those confident and able to deploy rapidly, this framework would allow them to do so. For the more risk-averse developers, this framework would allow the implementation of checks and gates to placate each stakeholder. While this does not solve the inherent nature of risk aversion in continuous development, it allows stakeholders to adopt continuous practices at their own pace, unrestrained by those adopting at a slower pace.

This solution is similar in nature to the concept of fine grained access control,

which allows administrators to determine the action of any user on any given stage within the pipeline.

4.4.3 Lack of Comprehension

Lack of comprehension is discussed repeatedly in this manuscript. While official definitions for each of the continuous practices are still in dispute, I did not feel it was inside the scope of the research to definitively define each term, and associated processes. However, with so much evidence pointing to a clear lack of understanding of continuous practices, I did feel it was prudent to select lack of comprehension as a critical challenge to be addressed by this research. Several design requirement solutions were prototyped to address this challenge. Having consistent use of terminology across an entire offering was suggested and implemented with IBM Cloud. Looking into the tools themselves, I found that determining the exact development process, from check-in to deployment and monitoring, was extremely difficult to determine in more complex systems. Based on suggestions from exploratory interviews, a complete end-to-end development overview located in a central place would improve users' ability to understand whichever deployment process their organization is using.

4.4.4 Lack of Early Feedback Implementation

It has been well established that one of the most important benefits from continuous development is early feedback, both from users and from the system. It embraces the notion of fail early and fix forward. However, over the course of my investigation, I noticed a critical component—collecting feedback and incorporating it into product development—was missing. Creating and sustaining the infrastructure that allows you to deploy continuously is the proverbial stick of continuous development while receiving early feedback is the proverbial carrot. Yet it appears most organizations are barley scratching the surface of feedback collection. By not utilizing feedback, organizations are paying the price of continuous without reaping any of the benefits. An analysis of IBM's Toolchain pipeline revealed one possible cause: there was no easy way to collect, organize, and display any form of feedback for developers. This problem was unique in the sense that it did not have an easily identifiable design requirement to solve it. As such, going forward it was determined I would seek out empirical data on the collection, analysis, and implementation of feedback by organizations using continuous development in the hopes a solution could be derived.

Challenges	Problem	Design Requirement
Lack of confidence - technical aspect	Manual Operation	Implementation of **control gates** to increase confidence to remove manual operations.
	Hesitation to release continuously	Optimize build time through **parallel test stage execution** reducing rollback time in cases of failure.
Lack of confidence - Cultural Aspect	Manual Operations	Remove manual operations for specific developers through implementation of **fine grained access control** within stages of the pipeline.
Lack of Comprehension	Apathy towards project development	Create an **interface allowing each member of an organization to view end to end development process.**
	Confusion on continuous development practices	Have **continuous development terms and definitions remain consistent** across an organizations offering.
Lack of Early Feedback Implementation	Users unable to benefit from continuous development	Obtain empirical data in order to devise solution.

Table 4.3: Results from Problem Characterization. Challenges defined and proposed design requirement solutions overcome them

Chapter 5

Artifact Design and Evaluation

5.1 Design Requirement Creation

Chapter 4 - Problem Characterization concluded with the determination of four continuous development adoption challenges: technical lack of confidence, cultural lack of confidence, lack of comprehension, and lack of feedback implementation. Each of these challenges directly affects the adoption of IBM's Continuous Delivery Pipeline. Chapter 4 also listed seven design requirement solutions to help overcome these four challenges: control gates to increase confidence and reduce manual operation, parallel test stage execution to reduce rollback time in case of failures, fine grained access control allowing organizations to reduce manual operations for specific developers, an interface overview of an organization's development process, consistent use of terminology and definitions, and obtaining data to create a solution that would allow organizations to incorporate feedback into their development process. Three of these design requirement solutions were incorporated into the IBM Toolchain at this point in the research project: consistent terminology usage, parallel test stage execution, and integrated logging. As such none of the three listen solutions were addressed by this thesis going forward.

As mentioned in Section 4.4, creating a solution for early feedback implementation required more input from continuous development practitioners. As such, no design requirements were initially developed. Three solutions remain: Control Gates, Fine Grained Access Control, and Interface Overview. Each of these 3 design requirements was iteratively developed working in collaboration with Mike Wilson and input from IBM UI/UX developers. We began by developing the logic for each requirement, how

it functions, how it is interacted with, how it interacts with the system, and how it impacts use of the system. Accessibility, consistency with existing design, usability, and other considerations were made when creating the user interface designs for these solutions.

I developed the design requirements through brainstorming with my industry collaborator IBM, with a heavy reliance on the results from the problem characterization steps. I would first discuss the functionality, design, or implementation of each component of a given design in a brainstorming meeting. Between each meeting, I would create a whiteboard mock-up with use cases for that component. During the next meeting, we would revise the component's design and the process would begin anew. Once my industry collaborator and I were confident the component was satisfactory it was created digitally using HTML to create a higher fidelity mock-up. This process continued for each design requirement until I reached a point where I felt they could no longer improve without further validation of the design. At this point, I conducted interviews and surveys to gather this feedback. Each section in this chapter represents one of those milestones.

Upon creation of the Interface Overview design requirement, we realized there already existed a platform to display a partial deployment overview, from source control leading up to the start of the pipeline. Instead of duplicating this work, I decided the design requirement would be created to become an addition to this existing framework. As it was then solely focusing on displaying deployment related information, such as traffic routing and running application information, this design requirement was renamed Deployment Methodology Overview.

Early User Interface Design Prototypes

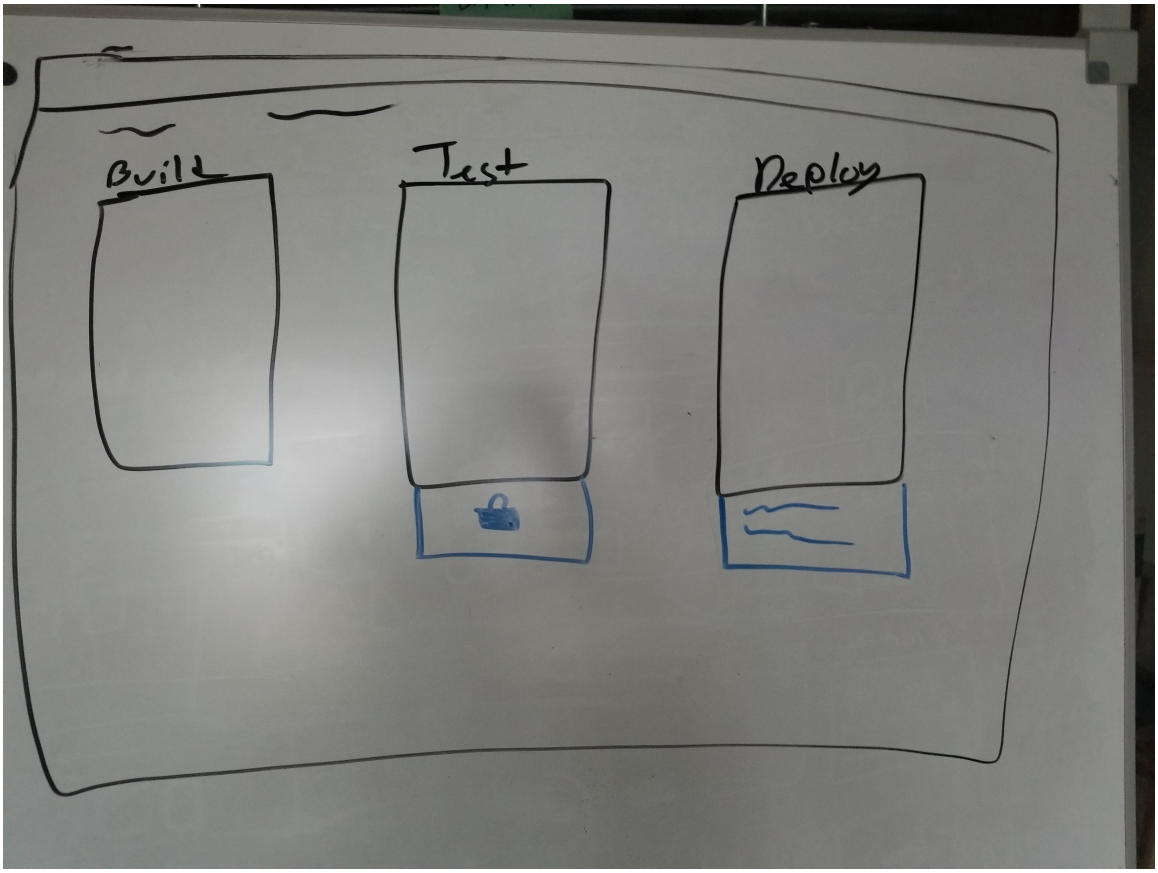

Figure 5.1: Early Control Gate Design Concept

Figure 5.1 shows an early *control gates* concept UI design. The initial design had *gates* acting as stop checks at the end of stages preventing the pipeline from progressing unless *gate* criteria were met. As seen in Figure 5.1, attached to the bottom of the Test stage is an additional box with a padlock icon inside of it. This would indicate the presence of a *gate* on the test stage. The initial design concept was such that users would navigate through these *gates* to identify the restrictive criteria that has been added to the stage the *control gate* is attached to.

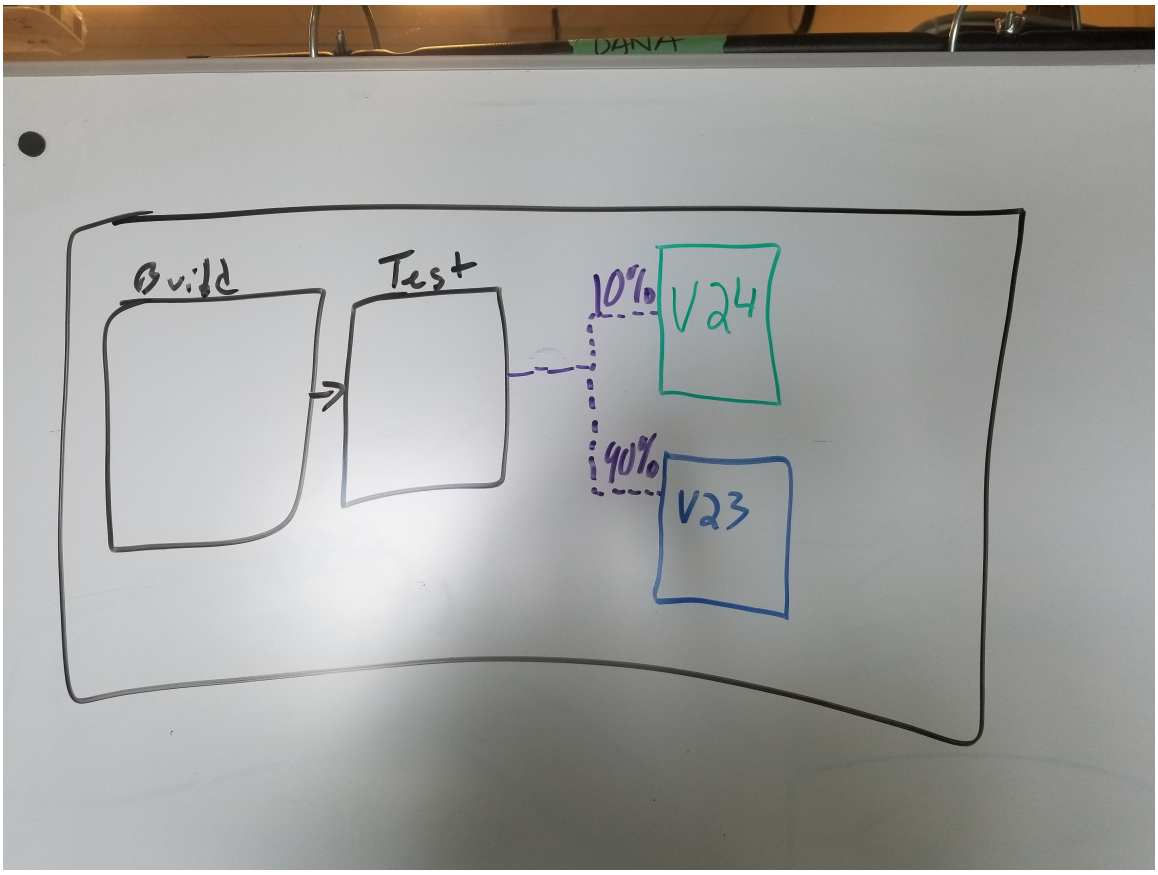

Figure 5.2: Early Interface Overview Design Concept

Figure 5.2 shows an early Interface Overview concept UI design. Upon creation of the Interface Overview design requirement, we realized there already existed a platform to display a partial deployment overview, from source control leading up to the start of the pipeline. Instead of duplicating this work I decided this design requirement would become an extension to this. As it was then solely focusing on displaying deployment related information, such as traffic routing and running application information, this design requirement was renamed Deployment Methodology Overview.

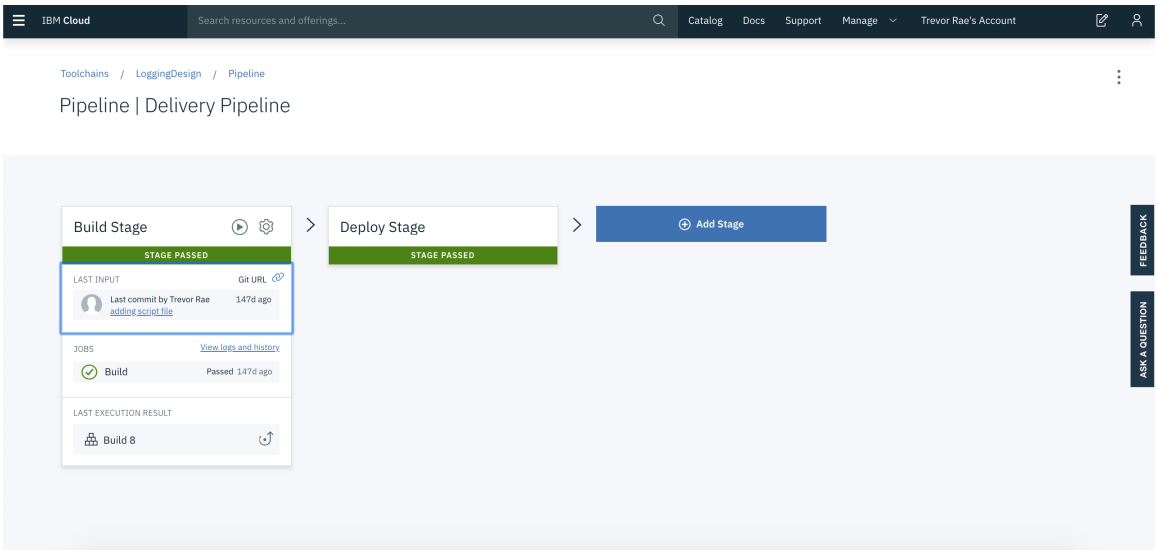

Figure 5.3: Early Fine Grained Access Control Design Concept

Figure 5.3 shows an early fine grained access control concept UI design. Fine Grained Access Control was the final design started, and as such, I began development using higher fidelity mock-ups as opposed to white board. Shown in Figure 5.3 is a user's view of a pipeline without view permissions for the deployment stage.

Each of these three design requirements were created to address one or more of the four salient challenges this thesis addresses. At each stage of artifact development, the four challenges were used as a guiding principal. Once I was satisfied that the three conceptual design requirements addressed the four challenges, I decided to validate the success of the artifact against the continuous software development community. This was done through a survey containing the early stages of the design requirements as seen above.

5.2 External Survey

5.2.1 Survey Design

Armed with the chosen set of adoption challenges and matching design requirements to address these challenges, a survey was created. The survey's objective was to validate our understanding of the prevalence of these adoption challenges, the impact on developers' day-to-day lives, and the design requirement solutions' ability to address these challenges. The survey was constructed in 2 sections, each attempting

to ascertain specific data. Section 1 was a series of general usage questions, inquiring as to the respondents' demographic information, tool usage, and development process. This information was collected in order to find correlations between tools, process(es), experience, and positions. Section 2 presented the artifacts to the respondent. Each of the artifacts was addressed in a section—and each section followed the same format. The artifact was presented and explained in a general sense, such that respondents who are unfamiliar with some of the terminology used in the survey can still understand the artifact and solution being discussed. The survey was deployed in various development communities in order to target as wide a range of respondents as possible.

5.2.2 Pilot

Prior to release to general public, the survey was piloted. The chosen pilot group consisted of four original interviewees, members of Mike Wilson's development team, as well as colleagues at SEGAL Labs. By obtaining feedback from this diverse group, it allowed the survey to benefit from academic and industry experience.

5.2.3 Results

The survey was distributed to the following sources with the listed responses:

Reddit /r/devops Responses : 24 — 140,000 Members

Reddit /r/CICD Responses : 8 — 683 Members

Reddit /r/samplesize Responses : 0 — 126,000 Members

Facebook - Hackathon Hackers Responses : 9 — 64,000 Members

Facebook - CdT Responses : 4 — 10,000 Members

Combined responses from all sources : 40

Aside

1. Data was reviewed as it came in to expedite the creation of the design artifacts.
2. 4 respondents chose not to consent and data was not gathered.
3. 2 Respondents submitted the survey twice, the duplicate data was removed.
4. 2 response sets were discarded as "fake" or misleading data.

Final responses : 32

Section 1

Demographic Information

The majority of respondents were senior developers, followed by junior developers and team leads A.1. Three-quarters of respondents have been in their position between 6 months and 5 years A.2. Over half had a team size of 10 or fewer active developers A.3. Most projects being worked on were services and web offerings A.4 being deployed to the public cloudA.5.

Development Tools

Github was the most popular tool used for source controlA.6 and had high popularity in planning and sprint managementA.7, issue trackingA.8, and testingA.9. Jira, and by extension Jenkins, had the largest useage for planning and sprint management A.7, issue trackingA.8, and pipelinesA.10. Both monitoringA.11 and testingA.9 had a number of tools being used, with no clear dominant tool emerging.

Measuring Level Of Continuous Development

Results show most respondents are able to propagate a code change through their development process, the practice of continuous delivery, in 1 hourA.12; however, to have code reach a state where it could be deployed, the practice of continuous integration, took between 5-15 minutes for the majority of respondentsA.13.

Pull request reviews the most prominent manual step for users staging their code before releasing into production A.1. There was a wide range of reasons given for lack of automation in production deploymentA.2.

If you indicated a lack of automation resulting in manual steps, please indicate any reason(s) these manual steps exist

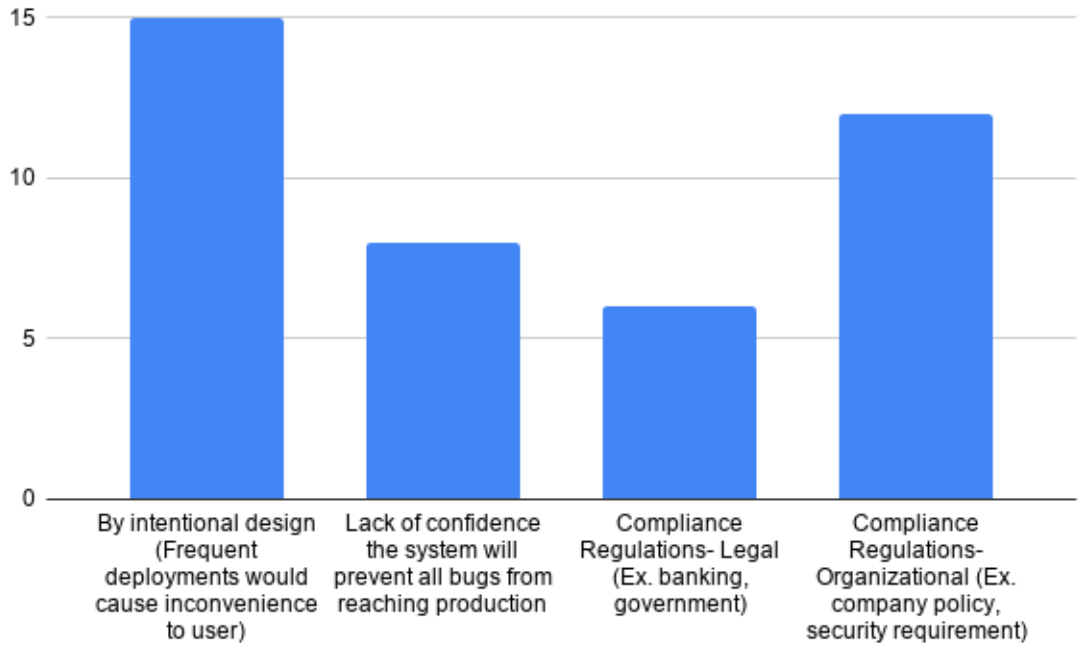

Figure 5.4: Reasons for Manual Steps

Section 2

Control Gates

Respondents were presented with the following description of a control gate: Control Gates regulate the flow of a pipeline. They typically exist as independent entities within the pipeline. Control Gates place requirements, such as test coverage, between stages in the pipeline. These requirements block the flow of the pipeline, halting progress until the requirements are met.

They were then asked if control gates would add value to their team's development practice on a scale from 1 - 5. Respondents overwhelmingly believe control gates would add value to their development practiceA.14. When asked to justify their response, most respondents indicated the increased testing capabilities was the primary value to be ascertained from control gates A.3 However, most respondents indicated the implementation of control gates would not increase their confidence to remove manual steps.A.15.

An interesting correlation can be found looking at the justification for control

gates not removing manual steps. The most common response was lack of connection between control gates and manual steps in the development processA.4. The majority of respondents who did not find a connection between control gates and removing manual steps also indicated they had manual steps in their deployment process by intentional design as indicated in Section 3.

Another interesting correlation is those users in Section 3 who associated manual steps in their deployment to a lack of confidence in their system all responded with neutral or positive to the implementation of control gates increasing confidence to removing manual steps.

Respondents Who Indicated Lack Of Confidence As Reason for Manual Steps In Deployment - Would Control Gates Increase Confidence To Remove Manual Steps

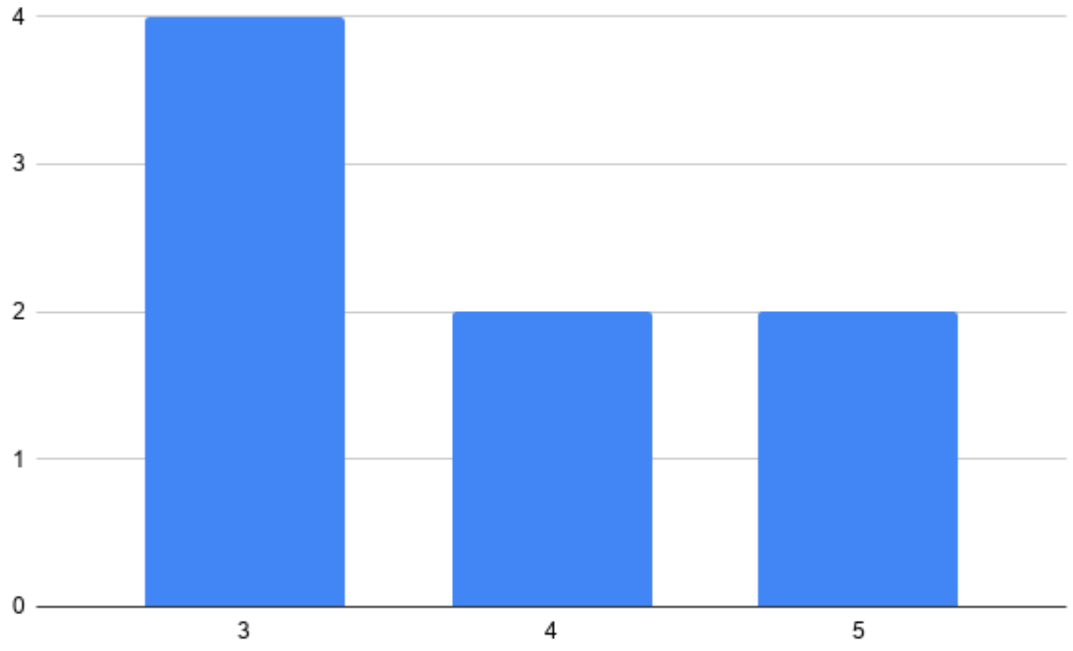

Figure 5.5: Users Who Felt Control Gates Would Remove Confidence And Who Also Indicated Lack Of Confidence In Removing Manual Steps

Fine Grained Access Control

To gain an understanding of base line feeling towards continuous development respondents were asked if they believe all members of a development team should be able to push to production regardless of role or experience. While most said they did not agree with that the difference was only 4 votesA.16.

Respondents were then presented with the following description of Fine-Grained Access Control: Fine-Grained Access Control allows pipeline administrators to control access to and actions on, any stage per individual within their development process. They were then asked if fine grained access control would add value to their team's development practice. Similar to control gates, overwhelmingly respondents indicated they would gain value from the implementation of fine grained access control in their development processA.17, primarily as a means to allow easier control over development operations for senior members and pipeline administratorsA.5.

Deployment Methodology

Respondents were presented with the following descriptions of deployment methods, then asked to select which methodologies were used in their team's development process. The methodologies presented were selected bedased on prevalence in the continuous software development community as ascertained in the previous work as well as by suggestion from Mike Wilson.

Canary Deployment: Creating a duplication of your production environment within an updated system which can be tested against a small number of users. Similar: Incremental Roll Out, Phased Roll Out

Blue/Green: Maintaining two or more identical production environments. Users are routed to the stable environment (blue). When a code update is applied, it is pushed to the non-stable environment (green) to undergo testing. On test success, users are routed to updated environment (green) and the previous environment (blue) is used for next update. Similar: Red/Black, Hot Standby

On Deck: A facsimile of the production server is run internal to the organization. Updated code is grouped and tested on it. When certain criteria, such as time or amount of changes is met, internal environment is pushed to production environment.

Dark Deployment: Deploys services into production that process real traffic but have no impact on the running system. A splice is set up along the code path, asynchronously duplicated traffic runs against dark services, and interactions are

logged.

Dark Launch: Deploys services into production that cannot be accessed unintentionally. One such method is adding a clause in the URL as the only point of access. Ex. index.html routes to standard version, indexdarklaunch.html routes to dark launched feature.

Feature Toggles: Features not ready for production are hidden, typically though a framework of conditional logic. Additionally, a specific interface can be built to toggle these features on and off allowing for testing in production.

A.19 The method most used was Blue/Green Deployment, followed by canary and feature toggles A.19. Most respondents indicated they used multiple types of deployment methods in combination with each other.

Users were then asked to describe challenges and benefits encountered deploying in the method they indicated. This question was an open text field as to not limit users to selecting problems I had identified. Their responses are shown in (Table 5.1).

Method	Adv/Dis	Reason	#
Blue/Green	Adv	Enables Continuous Integration	2
		Minimize downtime on deployment	4
		Enables Continuous Testing while maintaining stable version	4
		Rapid, repeatable, automated	2
	Dis	Does not support parallel deployment	1
		Cost sustaining multiple prod servers	1
		Large scheme/architecture create complex rollbacks	4
Canary	Adv	Minimize downtime	2
		Produces quality feedback/Enables AB testing	4
		Enables in depth error testing	2
	Dis	Edge cases still make it to production	1
		Extensive deploy time (bottleneck)	3
Feature Toggles	Adv	Ease of use/implementation	4
		Able to test function to pockets of users	1
		Prevents disparities when deploying to production	1
	Dis	Difficulty testing features exiting in quantum states	1
On Deck	Adv	Enables final testing of code before production	2
	Dis	Challenging to actually create replica environment	4
		Downtime when deploying	1
		Extensive overhead to set up on deck environment	1
Dark Launch	Adv	Enables testing in production environment	3
	Dis	A lot of work to test specific functionality	1

Table 5.1: Rationale - advantage/disadvantage for each deployment methodology

Feedback - User

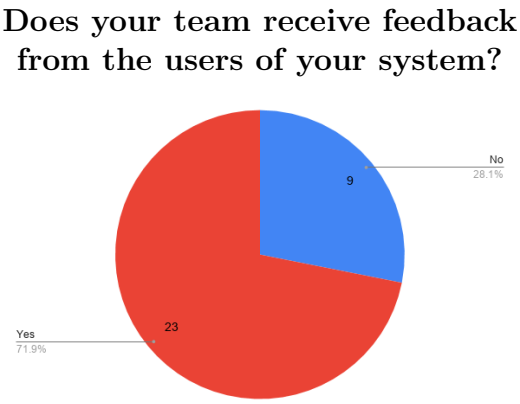

Figure 5.6: Receive User Feedback

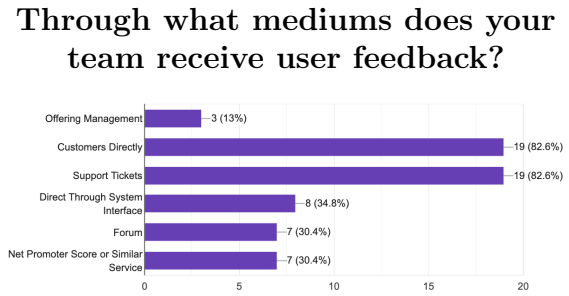

Figure 5.7: Medium for Feedback

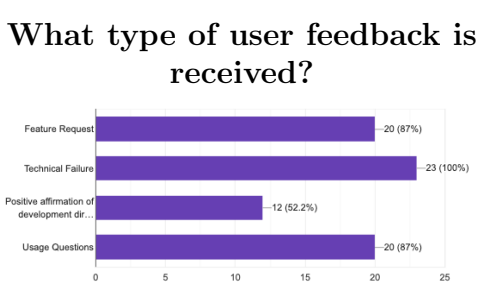

Figure 5.8: Type of Feedback

23 respondents indicated they received some form of user feedback 5.6. They were then asked through which mediums and what types of user feedback they received. Almost all respondents stated they received feedback directly from customers and support ticket 5.7. Every respondent indicated they received feedback on technical failures, with almost all also receiving usage questions and feature requests 5.8.

Respondents were then asked if they incorporate into their development 11 of the 32 respondents that indicated they received user feedback also claimed that some or all of the feedback received was used to steer development direction. Of those 11, 5 indicated they measure the results of implementing that feedback in their development to some extent. The predominant method of doing this was through Net Promoter Score feedback.

Feedback - System

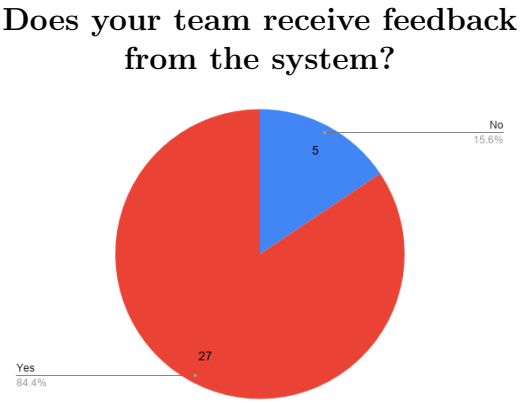

Figure 5.9: Receive System Feedback

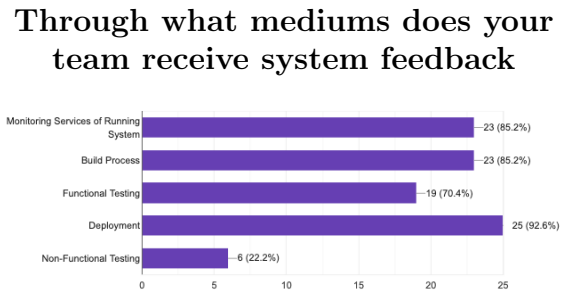

Figure 5.10: Medium for Feedback

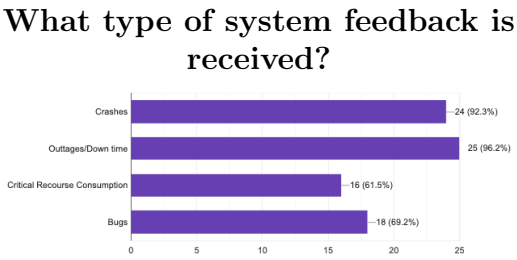

Figure 5.11: Type of Feedback

Over three-quarters of respondents received user feedback 5.9. Non Functional testing was the only method in which most respondents did not receive feedback from 5.10. Almost all types of feedback received were outages/down time and crashes 5.11.

10 of the 27 respondents that indicated they received system feedback also claimed that some or all of the feedback received was used to steer development direction. Of those 10, 3 indicated they measure the results of implementing that feedback in their development to some extent. One respondent indicated their team collected metrics such as delta over releases to analyze system performance.

5.3 Cross Section Correlations

Some interesting observations were made while correlating data points from different sections. Some of these correlations are be used in the result analysis done in Chapter 6 while others may be useful to note for future work. For example, 1 may suggest a link between development style for mobile applications and the ability to collect user feedback.

1) 100% of respondents who were developing mobile apps (Section 1) also collected user and system feedback (Section 7 and 8).

2) 8/11 respondents who deploy to the public cloud felt all members of a development team should be allowed to push to production.

3) 10/13 respondents who deploy to private cloud or baremetal felt not all members of a development team should be allowed to push to production.

4) 100% of respondents who had compliance regulations as a reason for manual steps in their deployment process also stated they did not feel comfortable allowing any member of their team to push to production.

5) 100% of respondents who had compliance regulations as a reason for manual steps in their deployment process also stated they didn't not think control gates would enable them to remove manual steps in their development process.

6) 100% of respondents who had compliance regulations as a reason for manual steps in their deployment process also stated they didn't not think fine grain access control would enable them to remove manual steps in their development process.

5.4 Survey and Follow Up Interviews

The external survey was instrumental in validating the design requirement concepts presented. Advancements to the design requirements were enacted upon utilizing feedback from the external survey. After exhausting this feedback, the artifact was in its final stage—a series of high fidelity video walkthrough detailing the use, functionality, and presentation of the design requirements. The next step was to present these video walkthroughs of the design requirements to IBM and the Continuous Development community in order to validate the implementation and receive feedback on design direction.

5.4.1 Survey Design

The UI implementation of the design requirements were displayed through a video walk through embedded in the survey. The video walkthrough consisted of a series of high fidelity images displaying the functionality, logic, and use cases for each of the three solutions. Like the design requirements themselves, these walk through videos were created iteratively with direct collaboration with Mike Wilson from the IBM Continuous Delivery Pipeline Team. The three design requirements presented in the survey included: Control Gates5.4.2, Deployment Overview 5.4.2, and Fine Grain Access Control 5.4.2.

5.4.2 Pilot

These designs were piloted. The pilot group included members of Mike Wilson's team along with colleagues at Segal Labs. By obtaining feedback from this diverse group it allowed the survey to gain insight from both academic and industry experience.

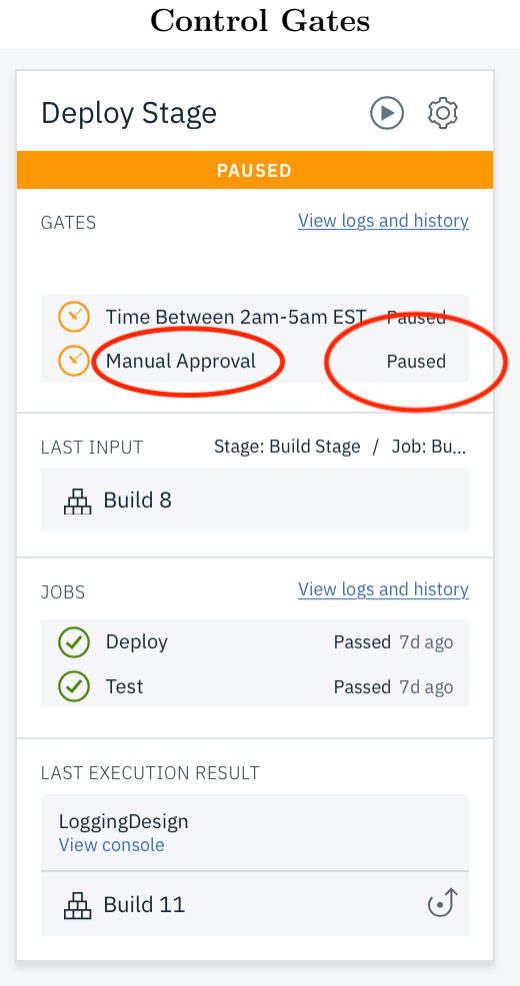

Figure 5.12: Deployment Stage with Control Gate added

Shown in **Figure 5.12** is high fidelity mock-up of a deploy stage for a pipeline with 2 conditional requirements enforced via a control gate. The first conditional being a timed gate requiring a specific time window, between 2am-5am EST, to be met before deployment can occur. The second, and highlighted criteria is manual approval, which requires an authorized user to interact with the stage for a deployment to occur. A full walk through of this design requirement can be found: https://www.youtube.com/watch?v=6IbsC8CA3Gs

Deployment Overview

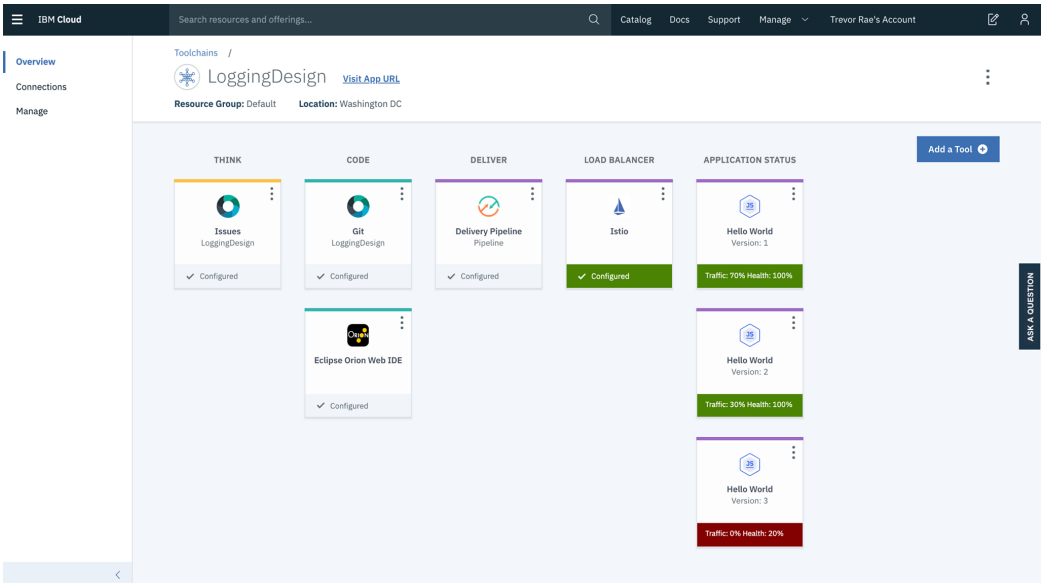

Figure 5.13: Deployment Methodologies

Figure 5.13 depicts the deployment overview artifact. It expands on the existing development suite (Think, Code, Deliver) to include operation status of live running applications (Load Balancer, Application Status). This additional content enables stakeholders to interact directly with various aspects of their running system. This includes front end services, multiple application versions, and multiple application environments. A complete walk through can be viewed here: https://www.youtube.com/watch?v=u4-jxXLrKKo

Fine Grained Access Control

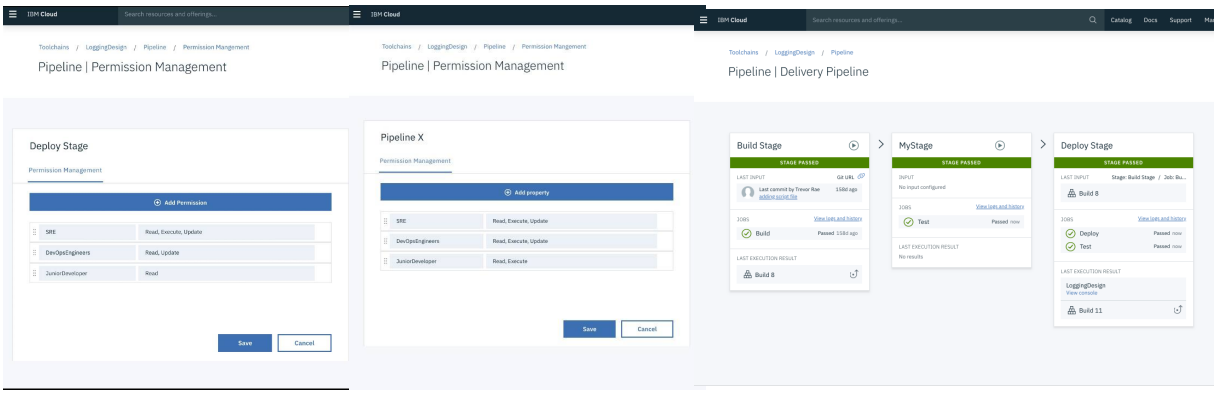

Figure 5.14: Permission Settings Control Users Pipeline Visual Interface

Allows pipeline administrators to control access to, and actions on, any stage per individual within their development process. **Figure 5.14** depicts permission control for a specific stage within a given pipeline. A full video explanation can be found at the following URL:
https://www.youtube.com/watch?v=B8wXFP21T-E

5.4.3 Survey and Follow Up Interview Results

As a final form of validation 10 additional individuals from IBM and the Continuous Development community were interviews/surveyed. Each individual was presented with the 3 design concepts and asked if:

(1) the core concept design would add value to their development process

(2) the design would enable them to adopt continuous development practices more easily through automation of manual processes

(3) if the specific implementation of the core concept met their needs/requirements

The results of these interview are shown in Table 5.2.

	Control Gates (#Respondents)	FGAC (#Respondents)	Deployment Methodology (#Respondents)
The core concept of the design requirement would add value to your development process (Yes/No)	10	9	10
The design requirement wold enable you to adopt continuous development practices more easily through automation of manual processes (Yes/No)	9	7	9
The specific design implementation satisfied user needs/requirements (Yes/No)	7	8	7

Table 5.2: Responses to interview/survey questions determining efficacy of design requirements presented.

Chapter 6

Discussion

This thesis began with the research question (RQ1): what are design requirements to improve the adoption of IBM's continuous development offerings. To achieve this goal, it must first be understood what challenges adopters of continuous development practices face (RQ2). Research question two was driven by the adoption challenges brought forward by IBM. Specifically, the challenges users face when assimilating continuous development. The solution implemented was the creation of technical design requirements for continuous development tools to improve the incremental adoption of continuous development processes. The model presented in Three Cycle View of Design Science [Hevner, 2007] was employed in the creation of these design requirements. Following this model, the discussion section will be split into 3 subsections: Problem Characterization and Development (See Chapter 4.4 - The Problem), Evolution of Artifact Creation and Validation, and Correlation with Existing Research. The Discussion will conclude with the contributions and limitations of this research.

6.1 Problem Characterization and Development

The first step towards creating design requirement solutions to aid in the adoption of IBM's Continuous Delivery Pipeline was to characterize the specific problem(s) that needed to be solved. This was done through many iterations of design science cycles including interviews, literature analysis, and industry collaboration. The result of this is discussed in detail in Section 4.4. Four salient problems were characterized: a lack of confidence stemming from the technology in use, a lack of confidence breed into the software development culture, lack of comprehension of the technology, terminology

and processes in the domain of continuous development, and finally an absence of early feedback implementation – a primary value gained from the use of continuous development.

6.2 Evolution of Artifact Creation and Validation

In addressing research question one, three artifacts were developed iteratively, validated and improved upon based on observations and feedback from both literature and the software development community—the target of this research. The three artifacts were design requirements addressing specific challenges identified in the problem characterization component of the research.

Control Gates were designed to address adopters lack of technical confidence in their development infrastructure to prevent outages and downtime in production. To safeguard against outages and downtime, organizations implement manual processes in their development process.

Fine Grain Access Control gives administrators/managers control to prescribe and delineate level and degree of access, for any user, on any stage of a pipeline. Fine Grain Access Control addresses the problem of cultural lack of confidence, the fear of allowing inexperienced members of the development team the same permissions as experienced members. This fear results in restrictions being applied unilaterally to all stakeholders in the development process.

The final design requirement was a **deployment overview**. Extending the static development overview to include dynamic run time information enables users to have a clear end-to-end overview of the deployment process. The challenge this addresses is a lack of understanding of the processes associated with continuous development practices for all stakeholders in the development team.

Result Analysis

Chapter 5 describes observations collected in experiments presenting these artifacts to various software development stakeholders. Respondents from the survey experiment, overwhelmingly believed both control gates and fine grained access control would add value to their development process. However, the supposition that this would empower stakeholders to remove manual steps from the development process was less favourable. Most respondents indicated that it would have no impact on their

removing manual processes from their development process. A closer analysis of the data revealed a possible explanation; approximately one-third of respondents disclosed that they had manual steps in the development process because of legal or organizational compliance regulation. All respondents subject to compliance regulations reported both control gates and fine grain access control would not enable them to remove manual steps. After eliminating respondents subject to compliance regulations, the vast majority believed that both fine grain access control and control gates would enable them to remove manual steps. The minority, being one respondent, believed strongly that these design requirements would in fact be detrimental to their development process. However, there was still a majority of users who felt that both control gates and fine grain access control would have no impact on the removal of manual steps. A possible explanation could be the nature of this experiment. While being a data validation step, the survey was also used to collect feedback on the design concepts presented. As such, respondents were only presented with one or two possible implementations of control gates and fine grain access control, hoping that they would provide detailed descriptions of how they would use these two features. Not providing more use cases could have lead to the negative feedback. A possible explanation is that respondents were not able to make the direct connection with the feature and the manual steps they would help to overcome. This argument is strengthened by the results from section 5.4, where users were shown a complete walk through of each design implementation with examples. In this case there was substantial support for control gates and fine grain access control removing manual steps from the development process.

6.3 How Design Requirements Overcome Adoption Challenges

Each of the design requirements created in this thesis was meticulously designed to address the adoption challenges identified during the problem characterization stage of the research. The three adoption challenges that have been detailed include lack of technical confidence, lack of cultural confidence, and lack of comprehension. This section will go over how each of the design requirements addresses these challenges.

6.3.1 Control Gates

Control gates had the pleasure of being one of the technical design requirements that were requested as well as having its core functionality address the lack of technical and cultural confidence as a by-product. The increased breadth of technical controls that can be implemented by control gates meet with many of the concerns about the pipeline's ability to limit errors in production. As a by-product, this also helped to improve cultural confidence as fear of deploying errors was diminished as well. Additionally, control gates have the ability to increase developer's quality of life as tasks such as releasing code at 2 am to minimize impact can now be automated instead of requiring one or more developers to manually do so.

6.3.2 Fine-Grained Access Control

Fine-Grained Access Control, like control gates, directly addresses both a lack of technical and cultural confidence. While using current pipeline technologies without fine-grained access control organizations are limited to coarse access control. The problem with coarse access control is organizations must limit all users of their pipeline to the restrictions placed on their least experience/trusted developer. These restrictions can create a negative culture in development teams. On the reverse, organizations that chose to enable their experienced developers by granting a wider set of permissions to all developers, including new developers, can create a lack of confidence for those new developers. From interviews, it was reported that joining a team where you immediately have access to the full range of permissions granted to their senior developers can produce a lack of trust in the system to catch mistakes which leads to a decrease in productivity. By implementing Fine-Grained Access Control organizations can circumvent these challenges. Assigning permissions to individuals at their comfort level. Increasing both cultural and technical confidence and enabling a more incremental adoption style that occurs at an individual level as opposed to team or organization-wide restricted progress.

6.3.3 Deployment Overview

Deployment Overview was created to primarily focus on the third adoption challenge - lack of comprehension. Through multiple steps in design science it became clear a large number of developers did not have a clear understanding of the difference

between the continuous practices and the processes associated with them. A lack of clear understanding of the full development process leads to miscommunication and a lack of code ownership, both of which are detrimental. By implementing a deployment overview into the pipeline it enables all those involved with product development to have a clear understanding of the path their product takes, for both development and operations activities.

6.4 Connection with existing research/literature

The value in assimilating continuous development practices is understood by the majority of the software development community. Excluding organizations impeded by compliance restrictions, the majority are making efforts to increase adoption of continuous development practices [Chen, 2017]. In Related Work (Chapter 2), various incremental assimilation methods of continuous development were discussed (Low Hanging Fruit, Value Driven Assimilation, and Pilot Team)[Stober and Hansmann, 2010] [Eck et al., 2014] [Svensson and Host, 2005] [Conboy and Wang, 2007] [Chen, 2017] [Chen, 2015]. In an ideal world, organizations would adopt all continuous practices overnight, however there exist a plethora of constrains limiting their ability to do so including lack of resources, lack of experience/expertise, organizational and legal compliance regulations, lack of confidence in their system or stakeholders, stakeholder conciliation, and a culture of fear of downtime/outages. Each of these constraints was identified in related work and reinforced by findings in my own research. The following discusses each constraint and the impact it has on continuous development assimilation.

Lack of resources was discussed in interviews. Lack of resources includes- personnel, time, and monetary resources. Multiple sources spoke of "Swamp". They defined swamp as development work that is required to appease stakeholders and maintain sustainability. My insight to this equated swamp work to technical debt that had been created and not dealt with. Additionally, swamp work is the result of organizational regulations passed onto a development team. Interviewees reported swamp work can consume anywhere from 40-80% of their development time. This lack of resources applies a heavy constraint on the ability to assimilate something as monuments as continuous development easily or quickly. Literature also heavily discusses the impact the number of resources spent on the assimilation of Continuous Development, in fact all new technologies, can have. When Chen (2017) was discussing how organizations

should use a pilot team to adopt Continuous Development he suggested organizations should select a product in which they care about a lot, and therefore are more willing to commit resources to. A case study conducted at Facebook and OANDA [Savor et al., 2016] concluded it is much easier for large organizations to adopt Continuous Development as such a significant amount of human resources are required to develop tools which are integral to adopting continuous development. Fitzgerald and Stol (2015) suggest in order for adopting to be successful, resources must be offered as needed as opposed to annual budget allocation.

This research project validated lack of experience/expertise in both interviews; 'we just don't know how' was an extremely common phrase, and developer forum analysis; the magnitude of traffic on posts seeking advice or clarification. While most understood the core principal of continuous development–rapid value delivery–the vast majority were baffled as to the precise steps to take within their organization to achieve this. In multiple case studies, including Paddy Power [Chen, 2017] [Chen, 2015] and two anonymous organizations [Olsson et al., 2012], the use of pilot teams is highly recommended to enable knowledge and experience transfer more dynamically within an organization.

With the recent spotlight on data privacy and security, organizational and legal compliance regulations have become a core component for most organizations. The survey in chapter 5 revealed some very interesting observations. When asked why manual steps exist in their development process over half of respondents reported there was some form of organizational and/or legal compliance blocking automation. Navigating compliance regulations can be difficult, and the penalty for failing to uphold it can be quite severe, as seen recently in the Google GDPR case [1]. The net impact of compliance regulations on the adoption of continuous development falls outside of the purview of this research. While compliance does affect continuous development adoption, the design requirements being created do not impact the regulations being levied at an organization. Future work in studying compliance role in continuous development has the potential to find solutions that would alleviate some of these adoption challenges. In their study of continuous development, Fitzgerald and Stol (2015) identified continuous compliance as an emergent practice.

Throughout the research project, a common question was asked in every round of interviews and surveys: why do you implement manual steps in your development practice? The previous constraint spoke of compliance restraints which was a common

[1] https://www.nytimes.com/2019/01/21/technology/google-europe-gdpr-fine.html

response. Others felt the resource cost was too high to implement automation [Savor et al., 2016]. Still more felt the pace at which they were developing was a good balance between safety and value gained. Deeper investigation identified a common theme, a software development culture of fear. Fear of downtime and outage, fear of having negative customer impacting events (CIE). This is true for all stakeholders in the software development process. Top-level stakeholders including managers, architects, and site reliability engineers designed their development process, specifically deployment, around minimizing CIE. This happened at the cost of automation and development speed. Developers were taught to fear creating CIE, causing a culture of risk aversion limiting development progress. This mentality runs counter-intuitive to assimilating continuous development practices. A significant amount of case studies reviewed for this thesis presented fear as a barrier to adoption. Some as the reason for failing to begin the adoption processes [Claps et al., 2015], others as a reason to spread out deployments as far as possible [Neely and Stolt, 2013]. However, there are others who state upon adoption, the risk associated with deployment decreases [Chen, 2017] [Chen, 2015] [Savor et al., 2016].

Stakeholder conciliation is an extremely prevalent theme found in this research and related works. There are a wide range of stakeholders involved in most software development and obtaining support from each stakeholder is critical [Savor et al., 2016] [Chen, 2017] [Neely and Stolt, 2013] [Fitzgerald and Stol, 2015] [Laukkanen et al., 2017]. Within my own research, it became quite clear the more senior the position of the person, the more their opinion affected an organization's ability to adopt. In my limited sample size, the teams who were least continuous had the most risk adverse senior stakeholders. Mitigating the affect of failing to obtain support from all stakeholders can overwhelm and organization and shackles continuous development assimilation speed.

For organizations attempting to adopt continuous development these constraints prevent this simple overnight transition. In these cases, incremental adoption has shown promise in enabling them to transition into continuous development practices at a pace that suits their development infrastructure while still satisfying which ever constraints they face. However, existing continuous development services are missing support for those organizations that wish to adopt continuously. Each of these design requirements enabled users to more easily adopt continuous development practices.

Stakeholder conciliation is a huge bottleneck for speed at which continuous development is assimilated. It some instances, it can even prevent adoption from occurring at all.

Previous work states obtaining top-down stakeholder support is the most critical, but all parties must support continuous development assimilation or else the adoption will be doomed to fail [Chen, 2017]. Each of the design requirements presented helps to address concerns raised by top-down stakeholders in the software development process. Deployment overview, in addition to its principal function, also conveniently provides a quick and easy means of access to the health of the running system as a whole, something top-level stakeholders have requested for quality of life improvement.

6.5 Contribution

This manuscript has two main contributions: an iteration on existing literature in the field of continuous development adoption and validated improvements to the IBM Toolchain and Pipeline offering.

6.5.1 Academic Contribution

Current academic research is being conducted on means to measure efficacy of continuous development, reduce adoption challenges, and exploring methods to improve the value gained from implementing continuous development, such as continuous experimentation. My work focuses on reducing challenges users face when adopting continuous development through industry implemented design requirements, and enabling adoption methods, such as incremental adoption, to be used by organizations. Future work can further build this collaborative effort with industry, using the means and methods described in this manuscript. Additionally, the empirical data collected can be utilized to draw correlations between the various adoption methods.

6.5.2 Industry Contribution

This research was coupled closely with industry needs. Tight collaboration with industry partners on all aspects of the research enabled vigorous relevancy cycles, guiding the research focus with real-time industry experience. The end result is three design requirements that will be implemented into continuous development services for widespread use. As is the nature of development, the specific design implementation can be iteratively improved upon in the future to continue to grow and evolve to fit the software development communities' needs.

6.6 Threats to Validity

Due to this research being primarily conducted with a singular research collaboration partner, IBM, the results may not have wide spread application and be generalizable. To address this, two iterations of the research were conducted external to IBM, analyzing forum posts and posting the survey to Reddit and Facebook, enabling external validation of artifacts developed in collaboration with IBM by the software development community.

Due to the small sample size, no statements about the statistical significance of these results can be made. Future work could provide a metric for determining the impact these design requirements had on the adoption of IBM's pipeline.

Appendix A

Additional Information

A.1 External Survey questions and results

What is your role within your team?

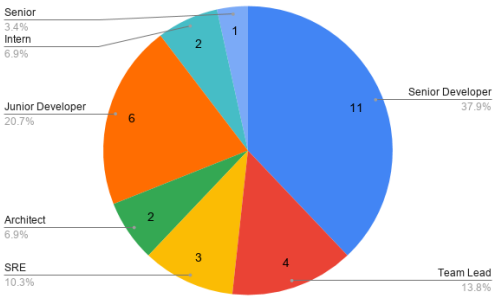

Figure A.1: Roles

How long have you been in your current position?

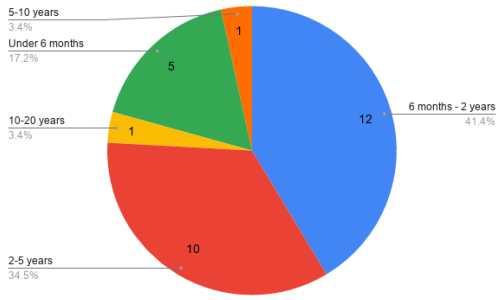

Figure A.2: Time In Current Position

Are there any manual steps from check in to a state where a code change could be deployed? If so, please list.	
Manual Steps	# of Respondents
Pull request review	19
Creation of release/branch promotion	4
Manual deployment approval	7
Manual testing	4
Quality Assurance team approval	1

Table A.1: Grouping of Manual Steps - Deployment Ready

Are there any manual steps from check in to live in production? If so please list?	
Manual Steps	# of Respondents
Manual branch promotion	6
Final code review	4
Quality Assurance final check	1
Environmental data manual update	1
Deployment script manually run	2

Table A.2: Grouping of Manual Steps - Live in Production

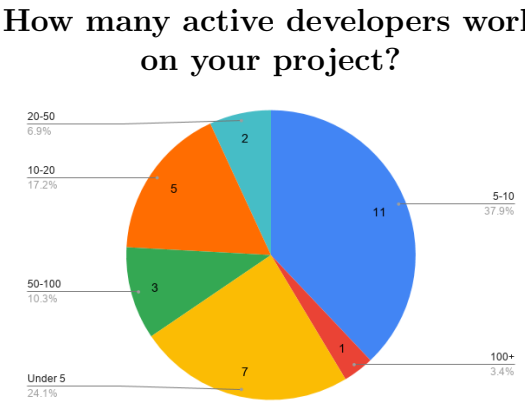

Figure A.3: Active Developers

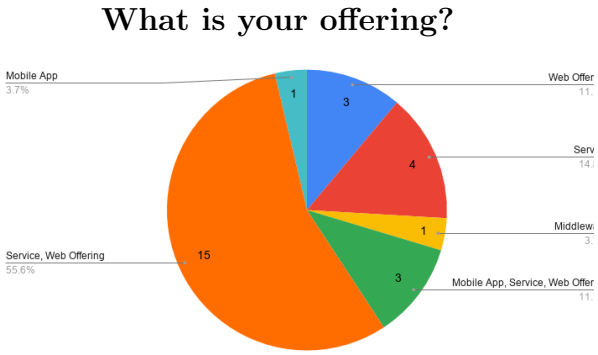

Figure A.4: Classifications of Offering

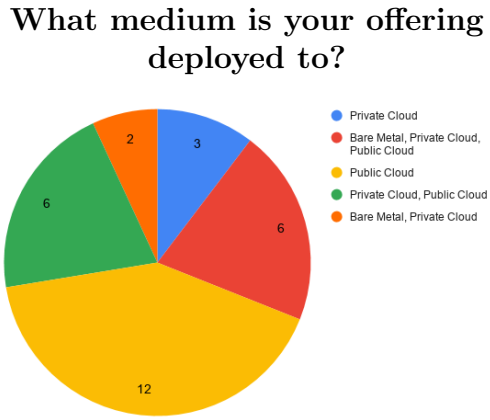

Figure A.5: Medium Product/Service Is Deployed To

Justification	# of Respondents
Respondents that indicated Fine-Grained Access Control would add value to their teams development process	
Increased ease to administer compliance regulations	3
Allow (senior members/pipeline administrators) to easily manage pipeline control	7
Easier to manage large development teams	2
Increase confidence in product quality	3
Respondents that indicated Fine-Grained Access Control would **not** add value to their teams development process	

All members of a team should have equal deployment rights	2
Respondents that indicated Fine-Grained Access Control would not impact their development process	
Process is already implemented through a technical work around (multiple pipelines/multiple repos)	1
Team is too small for this to have an impact	3

Table A.5: This table shows the range of justifications for responses to fine grained access control adding value

Justification	# of Respondents
Respondents that indicated Fine-Grained Access Control would increase confidence to remove manual steps from their development process provided the following justification	
Removal of manual steps for specific senior personnel	2
Respondents that indicated Fine-Grained Access Control would **not** increase confidence to remove manual steps from their development process provided the following justification	
Prevents developers from accessing resources they need	2
Respondents that indicated Fine-Grained Access Control would not influence their confidence to remove manual steps (neutral) from their development process provided the following justification	
No manual steps Fine-Grained Access Control would affect (Intentional/regulated/compliance)	6
No correlation between manual steps and Fine-Grained Access Control	3

Table A.6: This table shows the range of justifications for responses to fine grained access control removing manual steps

Justification	# of Respondents
Respondents that indicated control gates would add value provided the following justification	
Increase pipeline control	3
Increase confidence through additional testing	6
Increase in product quality	3
Stakeholder satisfaction	3
Policy Enforcement	2
Respondents that indicated control gates would **not** add value provided the following justification	
Developers should be responsible for their code deployment	1

Table A.3: This table shows the range of justifications for responses to control gate adding value

Justification	# of Respondents
Respondents that indicated control gates would increase confidence to remove manual steps	
Automation for manual approval at low traffic times	2
Respondents that indicated control gates would **not** increase confidence to remove manual step	
Implementation of control gates is a separate concern with manual steps (RW)	8
The manual steps in process are not affected by control gates	3
Legal/Organization compliance prevents removal of manual steps	3
Pull Request should always happen when pushing to production	3

Table A.4: This table shows the range of justifications for responses to control gate removing manual steps

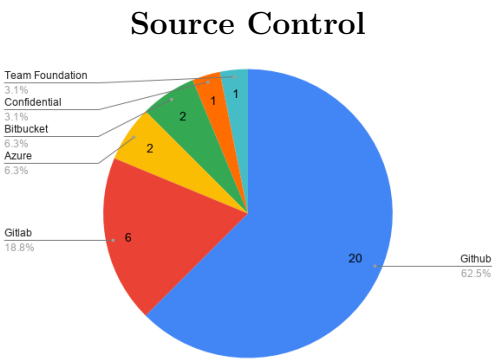

Figure A.6: Source Control

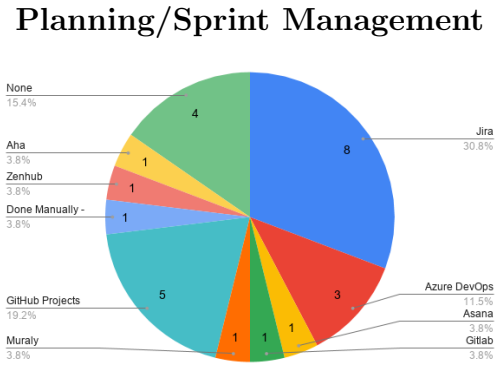

Figure A.7: Planning/Sprint Management

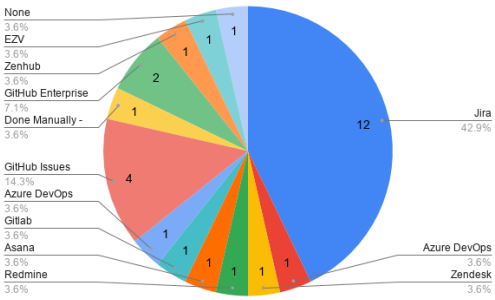

Figure A.8: Issue Tracking

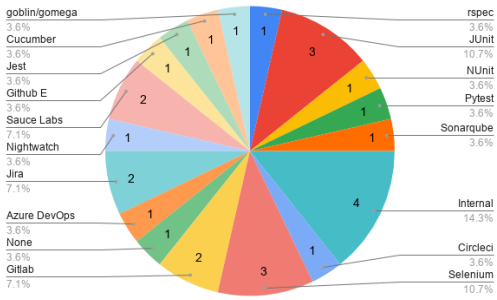

Figure A.9: Testing Suite

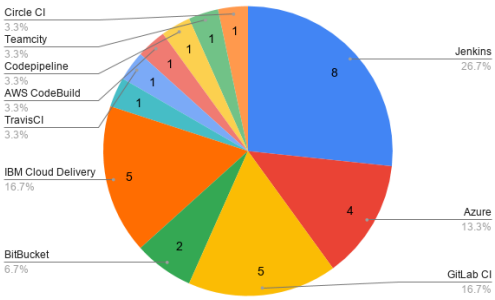

Figure A.10: Pipeline

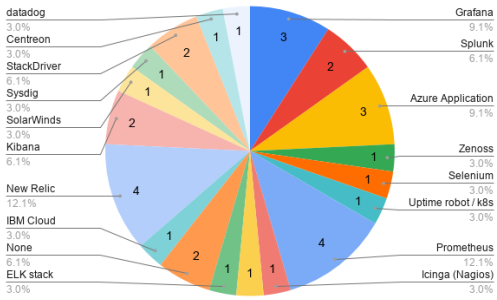

Figure A.11: Monitoring

Figure A.12: Fastest Time to Deploy

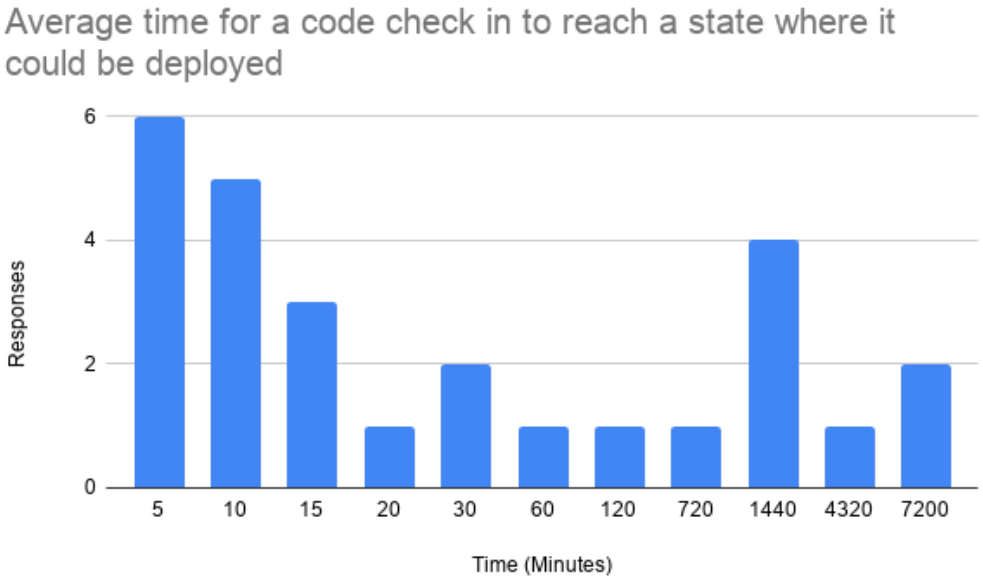

Figure A.13: Average Time to Deploy Ready State

Control Gates Would Add Value To Your Teams Development Practice

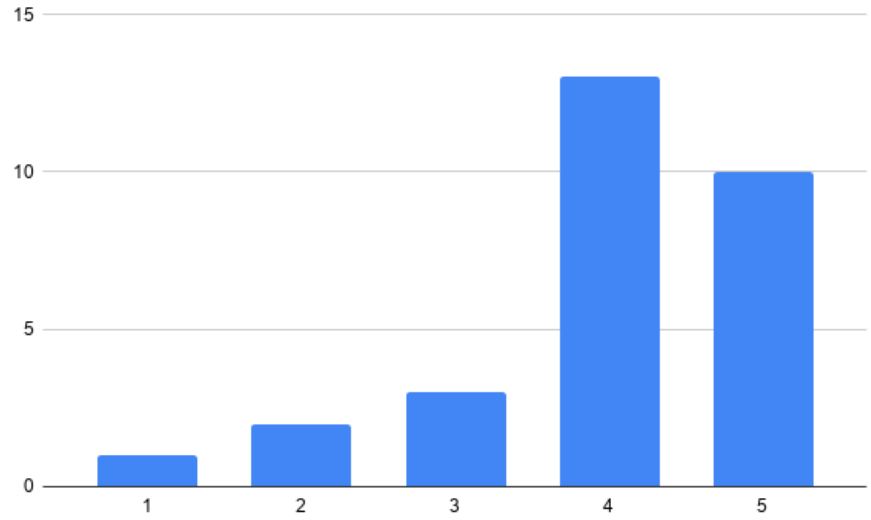

Figure A.14: Respondents Responses: 1 - 5 == Strongly Disagree - Strongly Agree

Control Gates would increase confidence to remove manual steps from your development practice

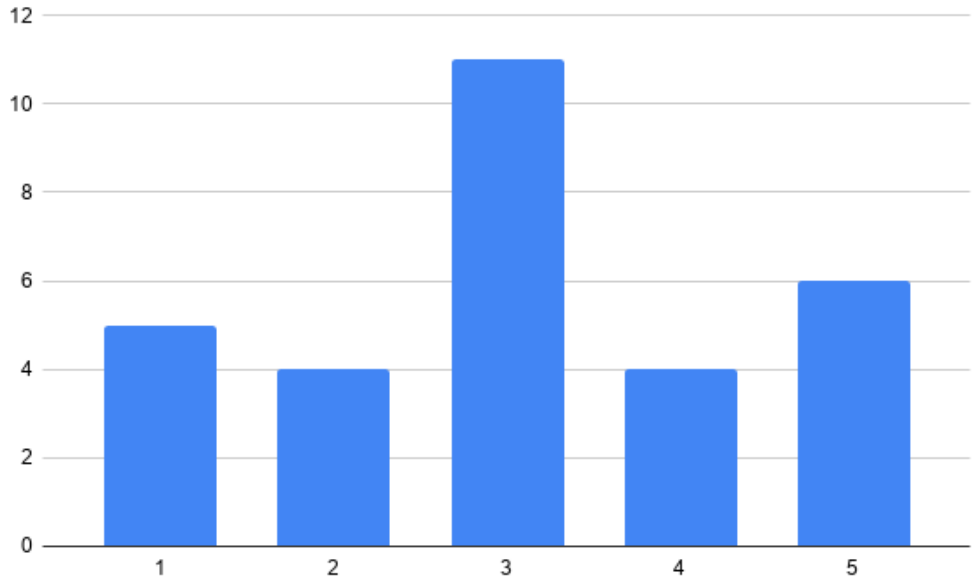

Figure A.15: Respondents Responses: 1 - 5 == Strongly Disagree - Strongly Agree

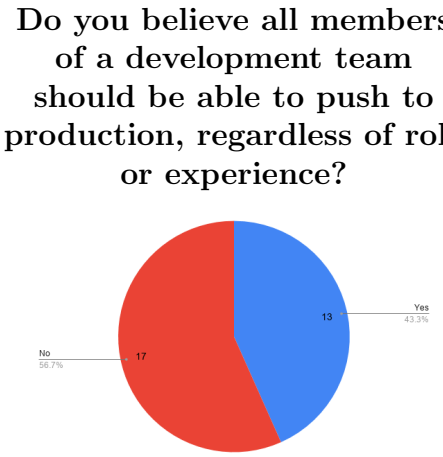

Figure A.16: Push to Prod

Figure A.17: Respondents Responses: 1 - 5 == Strongly Disagree - Strongly Agree

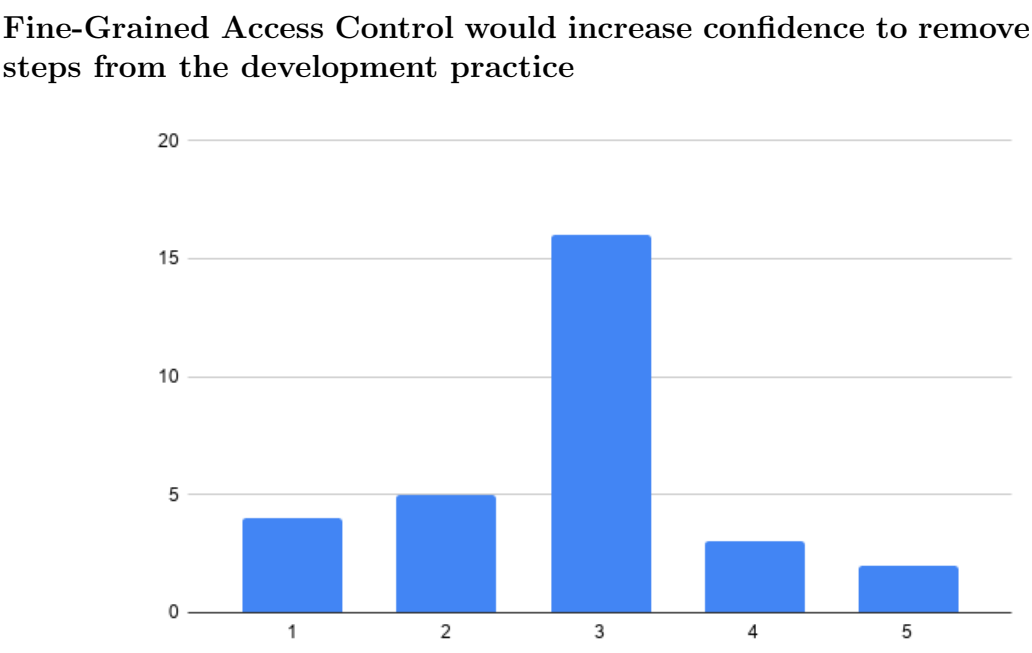

Figure A.18: Respondents Responses: 1 - 5 == Strongly Disagree - Strongly Agree

Which of the following methodologies best describes your teams deployment practice?

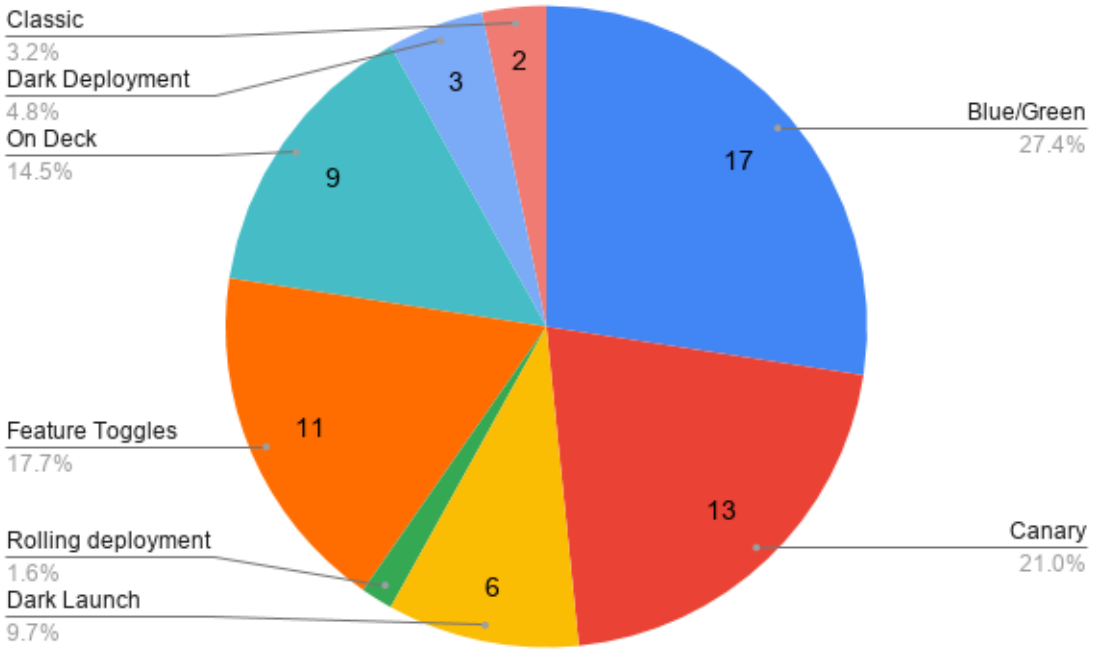

Figure A.19: Deployment Methodologies

Bibliography

Andi Mann, Michael Stahnke, A. B. N. K. (2016). state of devops report. *Puppet+ DORA*.

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., et al. (2001). Manifesto for agile software development.

Chen, L. (2015). Continuous Delivery: Huge Benefits, but Challenges Too. *IEEE Software*, 32(2):50–54.

Chen, L. (2017). Continuous Delivery: Overcoming adoption challenges. *Journal of Systems and Software*, 128:72–86.

Claps, G. G., Berntsson Svensson, R., and Aurum, A. (2015). On the journey to continuous deployment: Technical and social challenges along the way. *Information and Software Technology*, 57:21–31.

Conboy, K. and Wang, X. (2007). Agile practices in use from an innovation assimilation perspective: a multiple case study.

Dingsøyr, T. and Lassenius, C. (2016). Emerging themes in agile software development: Introduction to the special section on continuous value delivery. *Information and Software Technology*, 77:56–60.

Eck, A., Uebernickel, F., and Brenner, W. (2014). Fit for Continuous Integration: How Organizations Assimilate an Agile Practice. page 11.

Fichman, R. G. and Carroll, W. E. (2000). *The Diffusion and Assimilation of Information Technology Innovations*.

Fitzgerald, B. and Stol, K.-J. (2015). Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*, 123:176–189.

Forsgren, N., Smith, D., Humble, J., and Frazelle, J. (2019). 2019 accelerate state of devops report.

Fowler, M. (2016). Continuous Integration. page 14.

Hevner, A. R. (2007). A Three Cycle View of Design Science Research. 19:7.

Hevner, A. R., March, S. T., Park, J., and Ram, S. (2004). Design Science in Information Systems Research. *MIS Quarterly*, 28(1):75–105.

Humble, J. and Farley, D. (2010). *Continuous delivery: reliable software releases through build, test, and deployment automation*. Addison-Wesley, Upper Saddle River, NJ.

Laukkanen, E., Itkonen, J., and Lassenius, C. (2017). Problems, causes and solutions when adopting continuous deliveryâA systematic literature review. *Information and Software Technology*, 82:55–79.

Leppänen, M., Mäkinen, S., Pagels, M., Eloranta, V., Itkonen, J., Mäntylä, M. V., and Männistö, T. (2015). The highways and country roads to continuous deployment. *IEEE Software*, 32(2):64–72.

Montgomery, L. R. F. (2017). *Escalation prediction using feature engineering: addressing support ticket escalations within IBM's ecosystem*. PhD thesis.

Neely, S. and Stolt, S. (2013). Continuous Delivery? Easy! Just Change Everything (Well, Maybe It Is Not That Easy). In *2013 Agile Conference*, pages 121–128.

Olsson, H. H., Alahyari, H., and Bosch, J. (2012). Climbing the "Stairway to Heaven" â A Mulitiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software. In *2012 38th Euromicro Conference on Software Engineering and Advanced Applications*, pages 392–399.

Poppendieck, M. and Poppendieck, T. (2003). *Lean Software Development: An Agile Toolkit: An Agile Toolkit*. Addison-Wesley.

Ries, E. (2011). *The lean startup: How today's entrepreneurs use continuous innovation to create radically successful businesses*. Crown Books.

Rogers, E. M. (2010). *Diffusion of innovations*. Simon and Schuster.

Royce, W. W. (1987). Managing the development of large software systems: concepts and techniques. In *Proceedings of the 9th international conference on Software Engineering*, pages 328–338.

Savor, T., Douglas, M., Gentili, M., Williams, L., Beck, K., and Stumm, M. (2016). Continuous Deployment at Facebook and OANDA. In *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, pages 21–30.

Stober, T. and Hansmann, U. (2010). *Agile software development: best practices for large software development projects*. Springer, Berlin. OCLC: 489625465.

Ståhl, D. and Bosch, J. (2014). Modeling continuous integration practice differences in industry software development. *Journal of Systems and Software*, 87:48–59.

Svensson, H. and Host, M. (2005). Introducing an agile process in a software maintenance and evolution organization. In *Ninth European Conference on Software Maintenance and Reengineering*, pages 256–264.

Wieringa, R. (2009). Design science as nested problem solving. In *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology - DESRIST '09*, page 1. ACM Press.