Grounded Theory for DevOps Education

by

Candy Siu Tung Pang

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computing Science

University of Alberta

Abstract

DevOps, which stands for Development-Operations, is an important software engineering topic that arose from the IT industry in 2009. XebiaLabs claimed that, in Google Search, DevOps is one of the hottest search terms in technology over the last five years, and continues to rise [1]. According to Gartner in 2018, "DevOps processes would drive over 80% of the industrial-strength technology on the market" [2]. Currently, there is a high demand for DevOps practitioners, with many related job postings. DevOps is on its way becoming a required skill for IT practitioners. Therefore, I am motivated to study how people get their education and training to become DevOps practitioners, and how DevOps education should proceed in the future.

Since DevOps is very popular in industry, there are many blogs, webinars, conferences, communities, and organizations in industry trying to understand, promote, assist, and improve DevOps. However, there is limited academic research on DevOps. To the best of my knowledge, no research has studied the state of DevOps education.

DevOps education is different from other technical training, for DevOps is much more than a technology. DevOps is a cultural, procedural, and technological movement [3] [4] [5] [6] [7] [8] [9] [10]. Therefore, I employed Grounded Theory (GT), a social science qualitative research methodology, to study DevOps education. GT allowed me to discover the state of DevOps education from external data without making presumptions.

Following the GT process, I started with interviewing IT practitioners. Through GT's open coding and microanalysis, I discovered from the interviews that IT practitioners were interested in DevOps education and training. Continuing with the GT process, I used data analysis, interview, and survey to investigate the current state and challenges in DevOps education from academic and industrial perspectives. Based on my findings, I proposed future direction for DevOps education.

From the academic perspective, I found that most institutions were not teaching their undergraduate computer science students much about DevOps. Academics were not motivated to learn or adopt DevOps, for they thought the cost of adopting DevOps could not be justified in their research projects. There need to be the right incentives for academics to adopt DevOps, in order to stimulate interest in teaching DevOps.

From the industrial perspective, I found that industry has not clearly defined roles and responsibilities for DevOps practitioners. Therefore, it is not clear what students should learn to become DevOps practitioners. In addition, the DevOps community was evolving too fast for a single DevOps curriculum to be defined. Finally, DevOps experts established that classroom education and training was only the starting points of learning DevOps. DevOps practitioners advance their DevOps skills at work.

From my findings, I proposed five groups of future studies to establish the benefits academia stands to receive from DevOps education. The first group studies whether incorporating DevOps processes into programming assignments improves student learning outcomes. The second group studies whether DevOps skills are transferable to other computer science subjects. The third group studies whether DevOps adds value to academic publication. The fourth group studies whether DevOps knowledge improves students' employment opportunities. The fifth group studies how to align DevOps industrial practices with academic education. These future studies would lay the path and direction for DevOps education in academia.

The scope and impact of DevOps has expanded rapidly. In addition to DevOps education, I collaborated with other researchers to study other DevOps related topics. In one research study, I identified maintenance processes that are necessary to sustain a continuous environment in DevOps. In another research project, I defined interactions between microservices, which go hand-in-hand with DevOps. Finally, in a different research study, I found that programmers needed more knowledge about software energy consumption to support the DevOps' trend of cloud deployment. In conclusion, the field of DevOps is very wide. I have only studied DevOps education, its continuity, its processes interaction, and its effect on infrastructure. There are many more research opportunities in DevOps.

Preface

The research conducted on Continuous Integration (CI), Continuous Delivery (CR) and DevOps from Chapter 1 to Chapter 7 for this dissertation was done in cooperation with Dr. Abram Hindle and Dr. Denilson Barbosa at the University of Alberta. I am the lead collaborator of the research project. I have led the identification and design of the research program, the collection of data, the construction of surveys, the performance of interviews, the analysis of the research data, and manuscript composition. Dr. Abram Hindle has provided step-by-step guidance throughout the research project, and was involved in coding transcribed interviews and writing memos. During the Grounded Theory process, I am the primary investigator, and Dr. Abram is the secondary investigator. Dr. Abram Hindle and Dr. Denilson Barbosa have both contributed to data analysis and manuscript composition. The research project received research ethics approval from the University of Alberta Research Ethics Board, under project name "Continuous Integration", Pro00040635, July 26, 2013.

Chapter 8 of this dissertation has been published as Candy Pang, and Abram Hindle, "Continuous Maintenance", in the Proceeding of 2016 32nd IEEE International Conference on Software Maintenance and Evolution (ICSME), Raleigh, NC, USA, p.p.458-462. I was responsible for the problem identification, data collection and analysis, as well as the manuscript composition. Dr. Abram Hindle was the supervisory author, and was involved with concept formation and manuscript composition.

Chapter 9 of this dissertation has been published as Candy Pang, and Duane Szafron, "Single Source of Truth (SSOT) for Service Oriented Architecture (SOA)", in the Proceeding of 2014 12th International Conference on Service-Oriented Computing (ICSOC), Paris, Frence, p.p. 575-589. I was responsible for the identification and design of the research program, the collection of data, the construction of apparatus, the performance of experiments, the analysis of the research data, and the manuscript composition. Dr. Duane Szafron was the supervisory author, and was involved with concept formation and manuscript composition.

Chapter 10 of this dissertation has been published as Candy Pang, Abram Hindle, Bram Adams, and Ahmed E. Hassan, "What Do Programmers Know about Software Energy Consumption?", IEEE Software Magazine, May/June 2016, vol. 33, issue 3, p.p.83-89. Dr. Abram Hindle was responsible for the identification of the research problem, and the analysis of research data. I was responsible for the design of the research program, the construction of surveys, the collection of data, the analysis of the research data, and manuscript composition. Dr. Abram Hindle, Dr. Bram Adams and Dr. Ahmed E. Hassan were involved with analysis of the research data, concept formation and manuscript composition. The research project received research ethics approval from the University of Alberta Research Ethics Board, under project name "What Do Programmers Know About Powers?", No. Pro00041457, August 16, 2013.

IEEE is the publisher of the papers "Continuous Maintenance" and "What Do Programmers Know about Software Energy Consumption?". IEEE does not require individuals working on a dissertation to obtain a formal reuse license, but require the following message to be displayed.

Acknowledgement

Thank you Alberta Innovates Tech Futures (AITF) for providing 4 years of Alberta Innovates Graduate Student Scholarship. Thank you University of Alberta for providing 2 years of Queen Elizabeth II Graduate Scholarship - Doctoral level. Thank you Student Aid Alberta for providing Persons Case Scholarship.

Thank you Dr. Duane Szafron for encouraging me to come back for my Ph.D. program after leaving school for 13 years. Also thanks to Dr. Szafron for suggesting Dr. Abram Hindle and Dr. Denilson Barbosa as my supervisors in his retirement.

Thank you to Dr. Abram Hindle for his brilliant supervision. After returning from industry to the academic world, Dr. Hindle provided opportunities and guidance for me to explore research topics and methodologies that I had not worked on before. Since I was away from the academic world for an extensive period of time, Dr. Hindle provided extra support helping me to catch up with the latest academic research topics in software engineering. During my Ph.D. program, I experienced prolonged sickness, and Dr. Hindle was very patient during my recovery. Last but not least, thank you Dr. Hindle for the provision of funding.

Thank you to Dr. Denilson Barbosa for his supervision in preparing and defending my dissertation. Dr. Barbosa’s guidance during my dissertation writing has been highly appreciated. In addition, thank you Dr. Barbosa for the provision of funding.

Last but not least, thank you to Alan S. Koch, senior lead instructor of the DevOps certification course at ASPE Training, and Viktor Farcic, author of The DevOps 2.0 Toolkit - Automating the Continuous Deployment Pipeline with Containerized Microservices, for participating in our research.

Table of Contents

List of Tables

List of Figures

Glossary of Terms

AI - Artificial Intelligence

AWS - Amazon Web Services

BABOK - Business Analysis Book of Knowledge

CASCON - Centers for Advanced Studies Conference

CD - Continuous Deliver

CDSS - Collaborative Data Sharing System

CI - Continuous Integration

CM - Continuous Maintenance

COBIT - Control Objectives for Information and Related Technologies

COTS - Commercial Off-the-Shelf

CPU - Central Processing Unit

CRUD - Creation, Retrieval, Update, and Deletion

CSED - International Workshop on Continuous Software Evolution and Delivery

CVS - Concurrent Versions System

DBMS - Database Management System

DevOps - Stands for Development and Operations.

DevOps is about the culture, collaborative practices, and automation that aligns development and operations teams so they have a single mindset on improving customer experiences, responding to faster business needs, and ensuring that innovation is balanced with security and operational needs. [3].

DORA - DevOps Research and Assessment

EDIFACT - United Nations Directories for Electronic Data Interchange for Administration, Commerce and Transport

EPR - Electronic Patient Record

ESB - Enterprise Service Bus

ETL - Extract, Transform and Load

GPU - Graphics Processing Unit

GT - Grounded Theory

GUI - Graphical User Interface

HL7 - Health Level 7

ICE-DO - ICAgile Certified Expert in DevOps

ICP-FDO - ICAgile Certified Professional in Foundations of DevOps

ICP-IDO - ICAgile Certified Professional in Implementing DevOps

ICSE - International Conference on Software Engineering

IDE - Integrated Development Environment

IaaS - Infrastructure as a Service

IT - Information Technology

ITIL - Information Technology Infrastructure Library

ITSM - Information Technology Service Management

PC - Postal Code

PIM - Personal Information Management

PMBOK - Project Management Book of Knowledge

PMP - Project Management Professional

PaaS - Platform as a Service

QoS - Quality-of-Service

REST - Representational State Transfer

PHN - Patient’s Health Numbers

RDF - Resource Description Framework

ROI - Return on Investment

SE - Software Engineering

SLA - Service Level Agreement

SLR - Systematic Literature Review

SOA - Service Oriented Architecture

SSOT - Single Source of Truth

TDD - Test-Driven Development

TFS - Team Foundation Server

VM - Virtual Machine

WSDL - Web Service Definition Language

XP - Extreme Programming

YAML - YAML Aren’t Markup Language

1 Introduction

DevOps, which stands for Development-Operations, is one of the most important new movements in software engineering since 2009. Sacolick [3] defined DevOps as movement aligning development and operations teams in "organizational structure, practices, and culture to enable rapid development and scalable, reliable operations". XebiaLabs claimed that in Google Search, DevOps is one of the hottest search terms in technology over the last five years, and continues to rise [1]. It has been a popular topic within the Information Technology (IT) industry. According to a survey in 2016 [11], 80% of the organizations plan to adopt DevOps. In 2018, "Gartner estimated that DevOps processes would drive over 80% of the industrial-strength technology on the market" [2]. Currently, there is a high demand for DevOps practitioners. In 2019, according to a survey by DevOps Institute [12], 37% of the IT organizations are hiring DevOps practitioners, and 14% are planning to recruit within 12 months [13]. DevOps is on its way becoming a required skill for IT practitioners.

There are many blogs, webinars, conferences, communities, and organizations in industry trying to understand, promote, assist, and improve DevOps. However, there is limited academic research and curricula content about DevOps. If academics have limited knowledge about DevOps, then students are not likely to obtain the DevOps skills required by industry. Hence, students would not be able to fill the numerous DevOps openings. Therefore, I studied the current state of DevOps education, and identified the source of the challenges in improving DevOps education adoption. Based on my findings, I made hypotheses about how institutions might proceed to include DevOps in future education. To the best of my knowledge, no research has studied the state of DevOps education before. This is the first research that studies how to embrace DevOps education in academic institutions.

Unlike other software engineering research topics, DevOps is much more than a technology. Many IT practitioners argue that DevOps is a cultural, procedural, and technological movement [3] [4] [5] [6] [7] [8] [9] [10]. Therefore, learning DevOps is different from learning any other technology. Using the traditional quantitative scientific research methodology, I would have made hypotheses about DevOps education, then proved or dis-proved my hypotheses. Since there was no previous research about DevOps education, my hypotheses would solely be based on my presumptions, which I tried to avoid. Instead, I used Grounded Theory (GT), a social sciences qualitative research methodology. GT allowed me to discover the state of DevOps education from external data without making presumptions.

To fulfill the great demand of DevOps practitioners now and in the future, I want to find out how institutions can teach their students about DevOps. In this dissertation, I uses GT to study the current state of DevOps education from both academic and industrial perspectives. Based on the current state, this dissertation proposes methods to advance DevOps education that benefits academia and industry. The next two sections will summarize my findings of the current state of DevOps education and my proposals for future DevOps education advancement.

1.1 Dissertation Contributions

Following Grounded Theory (GT), the social sciences methodology, I started the project by interviewing IT practitioners. Then, through GT's open coding and microanalysis, I, in consultation with my supervisors, identified seven major categories of DevOps research topics, and 119 DevOps research questions. One of the categories is about DevOps training and education. Continuing with the GT process, I further investigated the current state and challenges in DevOps education from academic and industrial perspectives.

From the academic perspective, my first finding was that most institutions were not teaching their undergraduate computer science students about DevOps. I reviewed the undergraduate computer science curriculum of 50 prestigious engineering and technology institutions in the world. I studied their curricula, searching for different DevOps related terms to assess what DevOps topics institutions covered. The results show that most institutions did not cover much about DevOps at the time of writing this dissertation.

My second and third findings were that academics were not keen to learn or adopt DevOps. Even though DevOps benefits software development [5] [11] [14] [15] [16], academics would not adopt DevOps in research projects that involved software development. I invited academics to participate in a DevOps education survey and presentation, and I interviewed graduate students who develop software for their research projects. The graduate students all agreed that adopting DevOps could improve the quality of their software, and could allow others to reproduce their research results more easily. However, they felt that allowing others to reproduce their results would increase the replicability of their research, but would not significantly increase their chance of publishing. Hence, the cost of adopting DevOps could not be justified in research projects, and graduate students would not adopt DevOps. There needs to be the right incentives for academics to adopt DevOps, in order to stimulate interest in teaching DevOps.

From the industrial perspective, my fourth finding was that industry has not clearly defined responsibilities for DevOps practitioners, even though DevOps is popular and highly demanded. I analyzed the job requirements for DevOps practitioners. The employers expected DevOps practitioners to be Jacks-of-All-Trades. Job requirements in DevOps postings were totally different from posting to posting. It is not clear what students should learn to become DevOps practitioners.

My fifth finding was that the DevOps community was evolving too rapidly for a single DevOps curriculum to be defined. I examined DevOps certification availability [17] [18] [19], training opportunities [20] [21] [22], conferences [23] [24] [25] [26] [27] [28], communities [29] [30] [31] [32], and organizations [33]. The DevOps community was booming, but in the early stage of formalizing its credentials and defining its boundaries. Defining a curriculum for DevOps is like shooting a moving target; therefore, accredited DevOps curriculum is not available for education.

My sixth finding was that DevOps practitioners acquired their DevOps skills from work, not from formal education. It is not easy to learn DevOps within a classroom. I interviewed DevOps experts about their experience in learning DevOps. The experts confirmed that classroom training and education was only the starting point of learning DevOps. IT practitioners advanced their DevOps skills through hands-on working experience. Academic institutions need to figure out how to bring work experience into DevOps education.

In conclusion, even though DevOps is becoming a requirement for IT practitioners, there are many challenges in DevOps education. From the academic perspective, academics do not perceive sufficient incentives to be involved in DevOps; therefore, DevOps rarely appears in institutions' curriculum. Academics lack interest in DevOps, for they do not believe DevOps gives them enough benefits. From the industrial perspective, the DevOps communities are working on the DevOps learning path, while DevOps is evolving rapidly. Furthermore, the IT industry has yet to agree on the responsibilities of DevOps practitioners, which makes defining DevOps curriculum even harder. At the moment, IT practitioners can only learn DevOps through hands-on working experience. With all these challenges, there is no clear direction about where DevOps education should start.

1.2 Hypotheses about Future DevOps Education

To advance DevOps education, I identified 11 DevOps education hypotheses to help define the future direction. Research based on these hypotheses will establish what DevOps education should start teaching; how learning DevOps benefits students; and how to bridge the gap between academia and industry. The 11 hypotheses are in five groups.

The first group studies whether embedding different DevOps processes into programming assignments could improve student learning outcomes. Educators keep looking for better ways to teach computer science; if embedding DevOps processes into teaching improve student learning outcomes, then educators may be motivated to do so.

The second group studies whether DevOps skills are transferable. The hypothesis studies whether knowing DevOps improves student learning outcomes in other computer science subjects. If so, it should motivate more computer science educators to adopt DevOps.

The third group studies whether DevOps adds value to academic publications and initiates joint DevOps research with computer science academics in different specializations. The results may draw more attention from academics toward DevOps.

The fourth group studies whether knowing DevOps could improve students' employment opportunities and speed up integration at work. If so, the results should motivate students to learn DevOps.

The fifth group studies how DevOps can affect industrial procedures, and whether educators can bring those effects into academic education. These hypotheses include how on-the-job experience enhances DevOps learning; what are the differences between adopting DevOps culture and DevOps technology; and which project management methodology benefits more from DevOps. The results can make DevOps education more industry oriented.

Chapter 5 will describe and explain these 11 DevOps education hypotheses in detail.

1.3 Additional Contributions

The scope and impact of DevOps has expanded rapidly within a short period of time. Therefore, there are many gaps waiting to be filled. In addition to my contributions regarding DevOps education, I collaborated with other researchers to fill some of those gaps.

I studied and identified continuous maintenance processes missing from existing DevOps publications, and published these findings in the paper "Continuous Maintenance" [34]. With the continuous maintenance processes defined, all future DevOps advancement in culture, procedure, or technology should take maintenance into account.

While adopting DevOps, many companies also adopt microservices, since "microservices and DevOps go hand-in-hand" [35]. Microservice is an architecture that "encapsulat[es] portions of your application into self-contained services that can be individually designed, developed, tested, and deployed" [36]. Each microservice is a Single Source of Truth (SSOT). We defined the interactions between microservices, that were fundamental for microservices deployment, and published in the paper "Single Source of Truth (SSOT) for Service Oriented Architecture (SOA)" [37]. The paper defined standard operations and interactions between microservices. Based on the standard operations, development tool can be created to generate source code for microservices' interactions, which makes DevOps adoption easier.

Research shows that organizations that adopt DevOps are likely to deploy their applications in data centers [11]. Data centers have limited power supply. Similarly, mobile device, which has become an essential part of our lives, also has limited power supply. Therefore, all programmers should try to minimize software energy consumption. I studied programmers' knowledge about software energy consumption. In my paper "What Do Programmers Know about Software Energy Consumption?" [38], I concluded that programmers lack knowledge and awareness about software energy consumption. Since DevOps adoption has soared, programmers' skill to minimize software energy consumption is very important.

1.4 Dissertation Organization

The rest of this dissertation is organized as follows. Section 2.1 provides the history of DevOps. Section 2.2.1 describes the many blogs, webinars, conferences, communities, and organizations in industry trying to understand, promote, assist, and improve DevOps. Section 2.2.2 describes the limited academic research on DevOps. Section 2.3 provides an introduction to GT.

Chapter 3 presents how I employed GT to identify potential DevOps research topics. Among the identified topics, I selected DevOps education as the topic for further study. Chapter 4 demonstrates how I employed GT to study the challenges in DevOps education through interviews, survey, data collecting, and analysis. Chapter 5 lists my hypotheses about DevOps education. Chapter 6 discusses the limitations in my GT study. Finally, Chapter 7 concludes my GT study.

Chapter 8 includes my published paper "Continuous Maintenance" [34]. Chapter 9 includes my published paper "Single Source of Truth (SSOT) for Service Oriented Architecture (SOA)" [37]. Chapter 10 includes my published paper "What Do Programmers Know about Software Energy Consumption?" [38].

2 Background

As described in Chapter 1, DevOps is a popular topic in industry, but not in academia. This dissertation studies the state of DevOps education using Grounded Theory (GT). In this chapter, I will present the history of DevOps; show the popularity of DevOps in industry; as well as highlight the general lack of DevOps interest in academia. I will also introduce the Grounded Theory (GT) methodology used in this research project, and describe existing software engineering research using GT.

2.1 Continuous Integration (CI), Continuous Delivery (CD), and DevOps History

In the early days of software development, module integration usually occurred near the end of a project and often failed, which caused much frustration [39]. Continuous Integration (CI) is a practice that executes module integration frequently throughout a project lifecycle in order to avoid tension at the end of the project.

Most IT practitioners credited the CI concept to Extreme Programming (XP) [40]. In XP, Beck suggested that "developers should be integrating and committing code into the code repository every few hours, whenever possible". However, Beck was only proposing a CI concept, without any action plan.

In the late 90's, the IT industry, especially the enterprises with abundant resources, saw the necessity and value of adopting CI to deal with the fast-growing complexity in software development. In the book "Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People" [41], Cusumano *et al.* described how Microsoft adopted and benefited from CI by "do[ing] everything in parallel, with frequent synchronizations". Microsoft called the model synch-and-stabilize or iterative enhancement, but they actually put CI into action. At the time, IT practitioners also referred the synch-and-stabilize model as "mile-stone", "daily build", "nightly build", or "zero-defect" [42].

The first book that fully documented CI practices, "Continuous Integration: Improving Software Quality and Reducing Risk" came in 2007 [39]. Duvall *et al.* detailed the six CI tasks as:

1. Continuous Code Integration
2. Continuous Database Integration
3. Continuous Testing
4. Continuous Inspection
5. Continuous Delivery
6. Continuous Feedback

These six tasks focused mainly on development. In 2010, Humble *et al.* expanded the CI concept into Continuous Delivery (CD) in the book "Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation" [43]. Humble *et al.* described the CD process as a Deployment Pipeline, an end-to-end process from software development to delivery. The deployment pipeline allowed any version of an application to be delivered to customers automatically as desired. Humble *et al.* covered many CI and CD practices and tools in their book. It is still one of the most referenced books regarding CI and CD.

CI and CD cover the software development processes up until software release. Once the software is released, the operations team usually takes over the responsibility to operate the software. DevOps expands the continuous processes, CI and CD, from development to operations, and it represents a combination of Development (Dev) and Operations (Ops).

DevOps is about the culture, collaborative practices, and automation that aligns development and operations teams so they have a single mindset on improving customer experiences, responding to faster business needs, and ensuring that innovation is balanced with security and operational needs.

- Isaac Sacolick in "What is DevOps? Transforming software development" [3]

The DevOps workflow is commonly known to IT practitioners as depicted in Figure 2.1 [44]. The DevOps workflow includes all CI and CD tasks and processes. The CD process includes two parts, deploy and release. In Figure 2.1, CD is depicted as release. The name DevOps originates from the presentation "10+ Deploys per Day: Dev and Ops Cooperation at Flikr" by John Allspaw and Paul Hammond in 2009 at the O'Reilly's Velocity conference [45]. This presentation described how Flikr streamlined the development, release and operational functions. The title was then shortened to #devops on Twitter.

In addition to CI, CD, and DevOps, there are many other continuous processes, like continuous planning, continuous use, continuous trust, continuous improvement, continuous

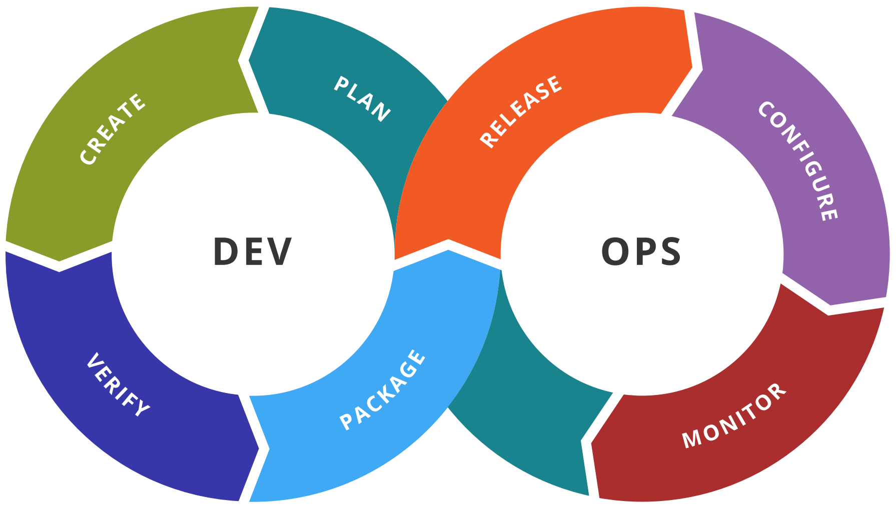


Figure 2.1 DevOps Toolchain [44]

innovation, etc. Fitzgerald *et al.* use the umbrella term "Continuous *" (continuous star) to identify the family of continuous activities [46]. There are also DevSecOps [47] [48] [49] [50], ChatOps [51], GitOps [52], DevOps 2.0 [53], and many more.

Technically, CI, CD, and DevOps are not the same. CD "is an approach whereby teams ensure that every change to the system can be released. … The goal of DevOps is to merge dev and ops roles together … to achieve business goals" [6]. However, most people in the IT industry, practitioners or not, use the term DevOps to cover all continuous processes. In the rest of this dissertation, we will use the term "DevOps" as an umbrella term to cover all continuous processes, but use "CI" or "CD" when referring to specific CI or CD processes. Throughout the project, I used the terms CI, CD, and DevOps in different surveys, interviews, and documents at the time. I kept those terms as-is in this dissertation in order to preserve the original content of the documents as they have been presented to the research participants.

As mentioned earlier, DevOps evolves from CI and CD. Therefore, up to now, more works were done on the Dev-side than on the Ops-side. Some IT practitioners expressed their frustration that "DevOps often can feel as though its real goal is to turn ITOps engineers into developers" [54]. When I started this GT project, my focus was on CI, since DevOps was not as popular then. Therefore, some of the earlier works have more focus on CI than DevOps.

2.2 Prevalence of DevOps

DevOps is a software engineering topic that arose from the IT industry. DevOps' popularity has increased among IT practitioners in the last five years [29], and caught a lot of attention from industry researchers [5] [11] [14] [15] [16] [55] [56] [57] [58]. XebiaLabs claimed that, during those five years, DevOps became one of the hottest technology search terms in Google Search, and it keeps rising [1]. However, the number of academic publications on DevOps is relatively low. In this section, we will compare the prevalence of DevOps in industry and in academia.

2.2.1 Prevalence in Industry

The popularity of DevOps in industry can be recognized in three ways: the positive publicity trends, the creation of many supporting tools, and the large number of surveys with many participants. I will describe these phenomena below.

2.2.1.1 *Evangelism*

Many DevOps communities, user groups, and blogs are created [29] to promote and inform IT practitioners on the topic. Even the most prominent industry technology research company Gartner Inc. "predicts that by 2020, half of the CIOs who have not transformed their teams' capabilities [to DevOps] will be displaced" [59].

Puppet [60] suggests, "DevOps practices lead to higher IT performance. This higher performance delivers improved business outcomes, as measured by productivity, profitability, and market share" (p.4) [16]. Therefore, DevOps has high return-on-investment value.

xMatters [61] declares that "DevOps practitioners are experiencing a faster time to market, improved customer experiences, fewer customer-facing incidents, faster resolution times, and a rapid-fire pace of innovation" (p.15) [9]. Therefore, DevOps gives companies a strong advantage over their competitors in the market.

Interop ITX [62], a technical conference organizer on the latest technical topics for Tech Leaders, added a DevOps Track to its conference [63]. Interop ITX recognizes that the "old waterfall approaches no longer serve the pace of today's application development and implementation cycles". Organizations are "dipping their toes into DevOps". Interop ITX also introduced DevOps certification program to its conferences [64].

All these organizations are praising DevOps and pushing the IT industry toward DevOps.

2.2.1.2 *Tools*

Since DevOps receives a lot of attention in industry, many companies and individuals are developing tools to support the Deployment Pipeline [43]. One of these companies is XebiaLabs [65]. XebiaLabs develops enterprise-scale DevOps tools, and they claimed that Forrester named

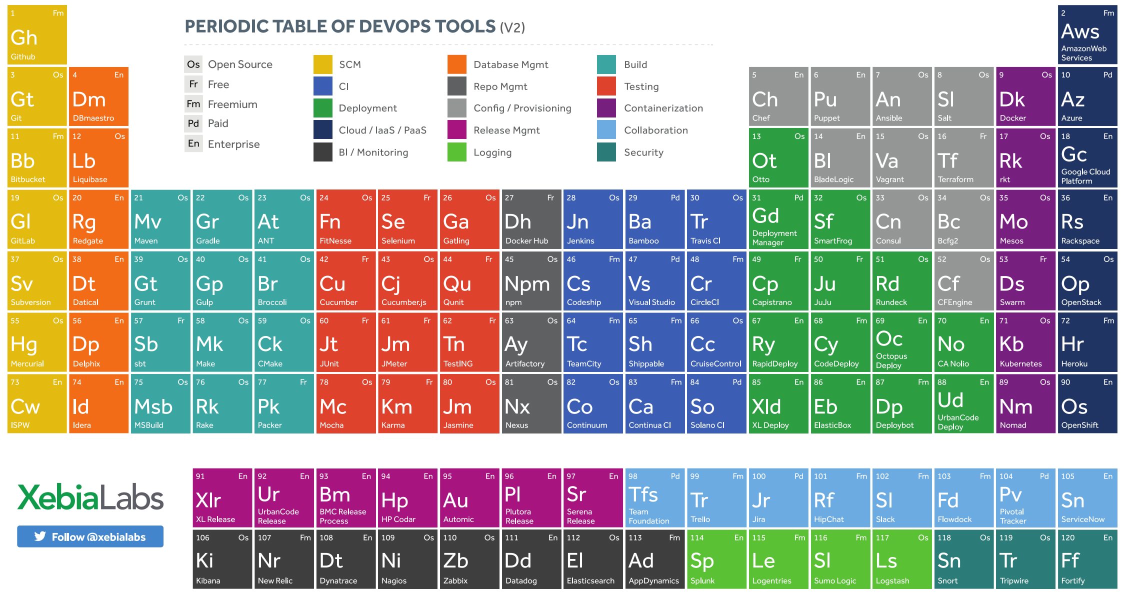


Figure 2.2 XebiaLabs Periodic Table of DevOps Tools (v2) [68]

them a leader in Continuous Delivery (CD) and Release Automation [66]. In addition, Gartner named XebiaLabs a leader in Application Release Automation [67].

XebiaLabs attempts to keep track of the ever-growing number of DevOps tools through the periodic table [68] shown in Figure 2.2. XebiaLabs classifies DevOps tools according to their licensing (Open Source, Free, Freemlum, Paid, and Enterprise) and the functionalities below.

1. Source Code Management
2. Repository Management
3. Database Management
4. Configuration Management or Provisioning
5. Build
6. Testing
7. Continuous Integration (CI)
8. Deployment
9. Release Management
10. Logging
11. Business Intelligence (BI) or Monitoring

    (Tools that generate insights and analytics from business data or system data)

12. Cloud, or Infrastructure as a Service (IaaS), or Platform as a Service (PaaS)
    (Tools that orchestrate shared infrastructure for applications deployment)
13. Containerization
    (Tools that create isolated user-space instances in an operating system)
14. Collaboration
    (Tools that collaborate all stakeholders in an application life cycle)
15. Security

The periodic table is evidence of the complicated DevOps ecosystem. On June 25, 2018, XebiaLabs released Version 3 of the Periodic Table of DevOps Tools [69]. Version 3 of the table includes 52 new DevOps tools and restructured functionality categories [70], which reflect the rapidly changing DevOps environment. XebiaLabs has also established an Ultimate DevOps Tool Chest with over 450 DevOps tools [71], and growing.

The large numbers of DevOps tools with different functionalities serve a wide spectrum of IT processes in DevOps. No single DevOps tool exists that supports all DevOps functionalities. However, some categories of DevOps tools are in the core of the process, and we will introduce a few of them below.

(#1) Source code management tools:

Source code management tools that manage source code revisions are the most important tool in DevOps. Git [72] is a popular distributed revision control system that allows for non-linear workflows. GitHub [73] is a web-based Git repository hosting service with its own added features. GitLab [74] is an open-source version of GitHub, with wiki and issue tracking features. Bitbucket [75] is another popular web-based hosting service for Git revision control systems, which offers free and paid services. No matter which product is used, software developers check source code into a source code management tool.

(#7) Continuous Integration (CI) tools:

CI tools retrieve source code from repositories, and execute pre-defined commands to build source code into applications. Travis CI [76] is a distributed web-based CI tool that builds projects hosted on GitHub and executes predefined YAML [77] build commands. Jenkins [78] is a server-based CI tool written in Java, and run in a servlet container. Jenkins supports many different types of source code management tools, and is able to execute different types of build commands.

(#13) Containerization tools:

Traditionally, a CI tool builds an application, then deploys it to an operating system. A containerization tool creates multiple isolated user-space instances in an operating system, so that multiple applications can be deployed to an operating system, to reduce resource requirements. Docker [79] is the first popular containerization tool. Docker is an open-source tool that automates the deployment of applications inside software containers. Kubernetes [80] is the most popular container orchestration engine [81]. Kubernetes is an open-source tool from Google that orchestrates deployment, scaling, and management of containerized applications across multiple hosts [82].

(#4) Configuration Management or Provisioning tools:

As described above, many different DevOps tools serve a wide spectrum of IT processes, and the tools are not connected. Therefore, there needs to be a configuration management tool that can configure and manage different system resources. Puppet [60] is an open-source configuration management utility, and it is one of the leading infrastructure-as-code solutions [1]. Puppet has multiple lines of DevOps products, such as Puppet Discovery, Puppet Enterprise, and Puppet Pipelines, that automate the modern enterprise. Puppet claims, from startups to "75% of the Fortune 100, thousands of companies rely on Puppet" [60].

Only a few categories of DevOps tools are described here, but there are many more. Not every DevOps pipeline uses all 15 categories of tools. Each pipeline adopts tools according to its needs. Therefore, there are many opportunities for new DevOps tool vendors.

2.2.1.3 *Surveys*

The large number of surveys about DevOps and the large number of IT practitioners answering those surveys confirm the popularity of DevOps in industry. I summarized some of the survey results below.

DZone's DevOps Zone [31] is one of the most active DevOps communities. Since 2014, DZone has been surveying IT practitioners on topics related to DevOps, and published annual reports. In 2014 [55], 2015 [56], 2016 [57], 2017 [58], and 2018 [5], DZone had surveyed 500+, 900+, 596, 497, and 549 technical professionals about DevOps respectively.

In 2015, the DZone survey reported 50% of the IT professionals "believed that they have implemented Continuous Delivery for some or all of their projects" (p.4) [56], but only 18% were certain that they had implemented the textbook version of CD. Hence, the majority of IT professionals have yet to catch up with the knowledge about CD. Nonetheless, 48% of those surveyed said that CD would become a universal IT standard practice. In 2017, "41% of respondents said their organization has a dedicated DevOps team" (p.3) [58]. In 2018, 48% of the organizations have dedicated DevOps teams. "DevOps has become … something that needs to happen sooner or later in order to stay competitive in a world where major companies like Amazon and Netflix can deploy changes every few minutes" (p.2) [5].

Puppet has published the State of DevOps Reports in 2017 and in five previous years, which documented the progressive DevOps adoption in the IT industry. In 2015 [14], 2016 [15], and 2017 [16] Puppet surveyed 4976, 4600, and 3200 technical professionals respectively. Puppet's surveys concluded that "DevOps practices lead to higher IT performance. This higher performance delivers improved business outcomes, as measured by productivity, profitability, and market share" (p.4) [16]. In 2016, these higher performing survey responders "are deploying 200 times more frequently …, and [has] the lowest change failure rate" (p.13) [15]. In 2017, these higher performing survey responders have "46 times more frequent code deployments; 440 times faster lead time from commit to deploy; 96 times faster mean time to recover from downtime; and 5 times lower change failure rate" (p.21) [16].

Interop ITX has surveyed 300 technology professionals involved in DevOps [11]. Interop ITX concluded from their survey that DevOps practices lead to reduced expenses and increased revenue. "Other benefits included increased collaboration, higher frequency of software deployments, a reduction in time spent maintaining applications, and improved quality and performance of applications" (p.4) [11].

All surveys come to the same general conclusion that DevOps is highly beneficial. Therefore, DevOps draws great attention from software development companies, and many try to adopt DevOps.

	CI	CD	DevOps	Container	Cloud	Deep Learning
IEEE Xplore						
In Title	(1998-2017) 80	(1994-2018) 26	(2014-2018) 62	(2000-2018) 1031	(2000-2018) 23097	(1999-2018) 1722
In Keyword	(2008-2017) 116	(2012-2018) 52	(2011-2018) 126	(1999-2018) 751	(2000-2018) 21596	(2006-2018) 1723
In Full-Text	(2000-2018) 1970	(2000-2018) 738	(2011-2018) 758	(2000-2018) 78109	(2000-2018) 145293	(2000-2018) 17716
ACM DL						
In Title	46	21	72	460	5286	353
In Keyword	67	33	71	279	3607	727
In Full-Text	1050	279	429	269521	32357	3672
Google Scholar						
In Title (Anytime)	509	456	607	188000	233000	11000
Since 2014	258	193	476	15100	32000	8630
Since 2017	83	54	158	4680	14700	4660
Since 2018	0	2	7	99	1070	400
In Full-Text (Anytime)	22300	27800	7650	3790000	3350000	159000
Since 2014	9410	8010	5240	224000	403000	287000
Since 2017	2770	2410	2000	23000	46200	19400
Since 2018	179	171	123	10900	18100	4050

Table 2.1 Academic Archives Search Results

2.2.2 Prevalence in Academia

Compared with its popularity in industry, DevOps does not receive much attention in academia. I show the lack of awareness by querying IEEE Xplore [83], ACM Digital Library [84], and Google Scholar [85] for academic publications on CI, CD, and DevOps. Next, I juxtapose those results against queries for currently trending topics such as containers, cloud computing, and deep learning. The results show that DevOps lack attention from academics.

In IEEE Xplore I searched for these exact phrases, "continuous integration", "continuous delivery", and "devops" in document titles, author keywords, and full-texts separately. In the ACM Digital Library, I searched with the same set of phrase in titles, keywords, and full-texts as well. In Google Scholar, I searched for these phrases in titles and full-texts. In Google Scholar I also tracked the trends of these phrase since 2018, 2017, 2014, and anytime. As comparisons, I also searched the phrase "container", "cloud", and "deep learning". I selected these comparison topics because they have gained their attention and popularity around the same time as CI, CD, and DevOps. Table 2.1 displays the search results.

On one hand, the phrases "continuous integration", "continuous delivery", and "devops" were very specific. The searches might not found articles using the earlier terminology such as "daily build" or "nightly build". On the other hand, the searches for the phrases "container", "cloud", and "deep learning" might return articles that were not related to software development or operations. Since I was only estimating the DevOps prevalence in academia, the leniency was acceptable.

From IEEE Xplore, CI first appears as a keyword in 2008, CD in 2012, and DevOps in 2011, while Microsoft had fully adopted CI in their development process by 1998 [41]. It took more than 10 years for academia to look into the CI, CD, and DevOps concept after it had become popular in industry.

Deep learning first appears as a keyword in 2006, near the first appearance of CI in 2008. However, by the end of January 2018, there are 1722 publications related to deep learning, while only 116 related to CI, 52 related to CD, and 126 related to DevOps. Obviously, CI, CD, and DevOps receive much less attention from academia. When a topic is the focus of a publication, the topic will usually appear in the title of the publication. While CI, CD, and DevOps have appeared in publication titles 46, 21, and 72 times, Container, Cloud, and Deep Learning have appeared in publications titles 1031, 23097 and 1722 times. CI, CD, and DevOps are less investigated academic research topics than Container, Cloud, and Deep Learning.

In ACM Digital Library, the publication counts are consistent with IEEE Xplore. The numbers of publications with CI (46), CD (21), and DevOps (72) in their titles are much less than those with container (460), cloud (5286), and deep learning (353) in their titles. While 727 publications identified deep learning as a keyword, only 67, 33, and 71 publications identified CI, CD, or DevOps as keywords respectively. The ACM Digital Library concurs with IEEE Xplore that CI, CD, and DevOps have not generated as much academic interest as container, cloud, and deep learning.

Erich *et al.* has performed a Systematic Literature Review (SLR) of DevOps publications in ACM Digital Library [86]. Erich *et al.* found 86 DevOps related academic publications between 2011 and 2016. The number is cohesive with my search result. By studying the academic publications, Erich *et al.* found that "the quality of the [DevOps] studies [was] quite low". DevOps may need to draw more attention in academic to provoke higher quality study. Similarly, Franca *et al.* also analyzed academic literature related to DevOps [87]. Franca *et al.* found that "DevOps [was] little explored by the academic community", which is cohesive with my finding.

Google Scholar further confirms that CI, CD, and DevOps have not received as much attention from academia as a research topic. Since 2014, publications on container, cloud, and deep learning are 10 times more than that of those on CI, CD, and DevOps. In the month of January 2018, there were zero, two, and seven publications on CI, CD, and DevOps, while there were 99, 1070, and 400 publications on container, cloud, and deep learning. The growing rates of CI, CD, or DevOps publications are also slower than container, cloud, and deep learning. Between 2014 and 2016, the average numbers of monthly publications on CI, CD, and DevOps were 4.9, 3.9, and 8.8. The average numbers have grown to 6.9, 4.5, and 13.2 in 2017, increased 42%, 17%, and 49% respectively. On the other hand, the average numbers of monthly publications on container, cloud, and deep learning between 2014 and 2016 were 289.4, 480.6, and 110.3. The average numbers have grown to 390, 1225, and 388.3 in 2017, with increases of 35%, 155%, and 252% respectively. Academic research on CI, CD, and DevOps is growing slower than other topics.

Practitioners in industry and researchers in academia pay different levels of attention to DevOps. They also have different research interests concerning DevOps. Garousi *et al.* compared industrial and academic publications on software testing, a subtopic of DevOps [88]. Industrial research focuses more on the actual practices, while academic research focuses more on theory. Garousi *et al.* also found that IT practitioners believed the academic publications were "too formal and hard to understand". Hence, IT practitioners do not consider academic research practical or useful. Therefore, industry-academia collaboration is rare.

Without the collaboration from industry, it is difficult to perform academic research on DevOps, as DevOps covers all software development processes performed by a board range of IT practitioners, including programmers, system administrator, DBA, operational analysts, support analysts, and more. DevOps also covers all infrastructure management for development, testing, staging, production, scaling, virtualization, and more. It is not likely that academia can recreate or simulate the complex industrial environment for research. Thus, industry-academia collaboration becomes very important. To show that a certain process or tool is making an improvement, academic researchers require cooperation and data from various participants in the deployment pipeline [43]. GT is a methodology that can effectively gather qualitative data from various participants. I will briefly describe GT in the next section.

2.3 Grounded Theory (GT)

Grounded theory (GT) is a qualitative research methodology. It is different from the traditional hypothetico-deductive approach. Its findings do not come from statistical procedures. GT uses "nonmathematical process of interpretation … to discover concepts and relationships in raw data and then organizes these results into a theoretical explanatory scheme" [89].

In 1967, Glaser and Strauss first described grounded theory in their seminal book "The Discovery of Grounded Theory" [90]. "The goal of GT is to generate theory rather than test or validate existing theory" [91]. Since 1967, many variants of GT have been published. GT is not only for social sciences, since it also "proved an extremely useful research approach in several fields including medical sociology, nursing, education, and management theory" [91].

2.3.1 Grounded Theory in Software Engineering

Software engineering research employed GT on many occasions. Stol *et al.* has compared various GT variants, and studied 98 software engineering publications that claimed to use GT [91]. However, Stol *et al.* found that "only 16 articles provide detailed accounts of their procedures" [91]. Most of these publications were "slurring" GT, hence claiming to use GT without actually following its guidelines or remaining compliant with the GT methodology. Based on suggestions from Stol *et al.*, I have followed the GT guidelines to create a compliant GT study on DevOps, and provide detailed accounts of my procedures.

I started my GT research by studying the two popular variants.

- Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory, Second Edition by Anselm Strauss and Juliet Corbin [89]
- Constructing Grounded Theory - A Practical Guide through Qualitative Analysis, by Kathy Charmaz [92]

The Charmaz version of GT includes repeating processes of coding (categorizing data for analysis - not programming), memo writing, theoretical sampling saturation, and sorting. I comprehended the constructivism in the Charmaz version of GT less than the post-modern theory in the Strauss and Corbin version of GT. Therefore, I selected to use the Strauss and

Corbin version of GT in my project, which had processes similar to the Charmaz version but was more methodological.

2.3.2 Strauss and Corbin Grounded Theory Summary

The Strauss and Corbin version of GT builds on the principles quoted below:

a. The need to get out into the field to discover what is really going on

b. The relevance of theory, grounded in data, to the development of a discipline and as a basis for social action

c. The complexity and variability of phenomena and of human action

d. The belief that persons are actors who take an active role in responding to problematic situations

e. The realization that persons act on the basis of meaning

f. The understanding that meaning is defined and redefined through interaction

g. A sensitivity to the evolving and unfolding nature of events (process)

h. An awareness of the interrelationships among conditions (structure), actions (processes), and consequences (p.9)

- *Anselm Strauss and Juliet Corbin in "Basics of Qualitative Research"* [89]

To satisfy these principles, I started my project:

- Without a predetermined theme or concept;
- Involving direct stakeholders from the fields;
- Investigating the goals and the interaction of the practitioners;
- Trying to identify the paths to the goals by identifying different aspects.

The Strauss and Corbin version of GT employs coding (open coding and selective coding), theoretical sampling, memos, conditional/consequential matrix, and diagrams. Strauss and Corbin use coding to conceptualize, reduce, elaborate, and relate concepts. They, then use non-statistical sampling, writing of memos, and diagramming to analyze the information. The Strauss and Corbin version of GT includes two phases as described below.

2.3.2.1 *The First Phase of Grounded Theory Project*

The Strauss and Corbin version of the GT process starts by gathering raw data from the stakeholders in the field, then the project investigators microanalyze the raw data through open coding.

Open coding definition: a process where investigators go through the raw data line-by-line to identify initial concepts, properties, and dimensions from the data.

Microanalysis definition: a process that examines the initial concepts, properties, and dimensions from open coding to generate initial categories, and studies the relationships among these categories.

Throughout the microanalysis process, investigators write memos to record ideas, concepts, thoughts, and deliberations during the analysis. At the end, investigators organize these concepts into discrete categories according to their properties and dimensions. Investigators will then associate these categories with potential theories, as Strauss and Corbin describe below:

At the heart of theorizing lies the interplay of making inductions (deriving concepts, their properties, and dimensions from data) and deductions (hypothesizing about the relationship between concepts, the relationships also are derived from data, but data that have been abstracted by the analyst from the raw data). (p.22)

- Anselm Strauss and Juliet Corbin in "Basics of Qualitative Research" [89]

Not all potential theories will be further developed. Investigators select a theory of interest to explore fully.

2.3.2.2 *The Second Phase of Grounded Theory Project*

The selected theory is formulated into logical, systematic, and explanatory dimensions. The dimensions are further developed and analyzed from many different angles. Investigators further examine each dimension through asking questions and making comparisons.

By asking questions, investigators gather additional data from stakeholders regarding specific dimensions of the potential theory from different perspectives, then analyze the data by selective coding (p.143) [89]. During selective coding, investigators go through the additional data to verify or reject the potential theory. The questioning and analyzing processes are repeated to refine the theory until theoretical saturation (p.143) [89].

Selective coding definition: a process of integrating and refining the theory.

Theoretical saturation definition: a point in theory development at which no new properties, dimensions, or relationships emerge during analysis.

Strauss and Corbin have not suggested a minimum number of shareholders to be questioned. "The ultimate criterion for determining whether or not to finalize the data-gathering processes from stakeholders is theoretical saturation" (p.158) [89]. If no new properties and dimensions emerge from the data, the data-gathering processes can stop even with a small number of samples.

Both selective coding and inter-rater reliability may have small sample size, but they have different goals. The participants in inter-rater reliability measurement are trained to provide rating according to predefined guideline to archive homogeneity or consensus. However, the participants in selective coding are urged to provide their free view, so that new properties, dimensions, or relationships can be discovered. The selective coding is saturated when new participants do not add new idea to the existing ones.

In addition to asking question, investigators further their examination by making comparisons. Investigators gather related data and existing results to verify or reject the potential theory through traditional quantitative scientific methodologies and conditional-consequential matrix (p.181) [89].

Conditional-consequential matrix definition: an analytic device suggested by Strauss and Corbin to stimulate thought about the relationships between macro and micro conditions/consequences, both related to each other and to the process.

Throughout the questioning and comparing processes, investigators write memos and diagrams to document thought and analyze results. Memos can take several forms, including code notes, theoretical notes, operational notes, and varieties of these. When writing the final report, investigators draw on the memos to back up the theory.

Finally, investigators may use theoretical sampling (p.201) [89] to verify the potential theory.

Theoretical sampling definition: a data gathering process driven by concepts derived from the evolving theory and based on the concept of making comparisons, whose purpose is to go to places, people, or events that will maximize opportunities to discover variations among concepts and to densify categories in terms of their properties and dimensions.

Only a short description of Strauss and Corbin grounded theory is included in this section. Please refer to "Basics of Qualitative Research" [89] for more detail.

2.4 Previous Software Engineering Research using Grounded Theory

Even through GT is a social sciences methodology, there is existing DevOps research using GT or GT's processes, but not on DevOps education. Adolph *et al.* used GT to study how individual aptitudes and team social factors affect project cost [93]. Using the GT methodology, Adolph *et* *al.* reported two theories. First, team training should be an essential part of software engineering education. Second, team leaders should have good leadership and facilitating skills, and respect their peers.

Hilton *et al.* employed interviews as a qualitative method to discover knowledge, then employed surveys to triangulate their findings [94]. Even though Hilton *et al.* did not mention GT directly, they straightly used GT processes to study developers' barriers and needs when using CI. Hilton *et al.* started by interviewing 16 field stakeholders to discover a potential theory. Then, they deployed two surveys, a focus survey with 51 participants and a broad survey with 523 participates, to verify the potential theory. The GT study concluded that developers had to make trade-off decisions between speed and stability; accessibility and security; flexibility and simplicity when using CI.

Hilton *et al.* also used a qualitative methodology to study the costs, benefits, and usage of CI in open-source software [95]. Hilton *et al.* concluded from their GT project that "developers encounter[ed] increased complexity, increased time costs, and new security concerns when working with CI" [95]. Therefore, academics have many opportunities to improve DevOps processes through industry-academia collaboration. To increase industry-academia collaboration, Garousi *et al.* suggested that academia should ensure "the research problems are based on real industry needs" [88].

Erich *et al.* used an approach inspired by GT to perform a Systematic Literature Review (SLR) of DevOps [86]. Erich *et al.* found that the quality of the DevOps academic studies is low, and "there exists little agreement about the characteristics of DevOps in the academic literature" [86]. Erich *et al.* continued their study in industry, and found that DevOps is a "personnel trait", a culture of automation. Similarly, Franca *et al.* used a GT inspired procedure to analyze a broad range of academic literature to define DevOps [87]. Franca *et al.* concluded that DevOps lacked consistent definition and evidence of benefits. Both of these publications claimed to be the results of GT projects. However, they did not actually employ the GT methodology, a phenomenon which Stol *et al.* opposed [91].

Luz *et al.* followed the Glaser and Strauss [90] version of GT closely to discover DevOps adoption theory [96]. They employed practitioner interviewing, open coding, selective coding, data analyzing, and theoretical coding to discover the success factors in DevOps adoption. Luz *et* *al.* presented a grounded theory that collaboration is the most important factor in DevOps adoption, more important than all other technical factors.

In conclusion, the evolution of DevOps goes from an iterative concept to continuous integration, to continuous delivery, then to DevOps. It goes from being large enterprises' practices to software developers' practices, to IT industry practices, and finally to an IT standard. The bright DevOps future inspires lively research and development activities in industry, but has not stimulated a strong number of publications in academia. The abundant DevOps research opportunities should be promoted to academics. To raise awareness about DevOps in academia, and encourage academic-industrial collaboration, my GT project identified DevOps research topics that interested the IT industry. I will describe how my GT project identified potential DevOps research topics in the next chapter.

3 Creation of Potential DevOps Grounded Theory

Grounded Theory (GT) is particularly suitable for my open-ended DevOps research. Many IT practitioners have identified DevOps as a cultural, procedural, and technological movement [3] [4] [5] [7] [8] [9] [97] [98], rather than only a technological innovation. Randall states that adopting DevOps technology without the corresponding culture would likely fail.

Tools are essential, but without a strong DevOps culture in place, tooling will only segregate, frustrate, and negatively impact any type of transformation work you're trying to accomplish.

- T.J. Randall in "Winning at Culture - The Keys to the Successful DevOps Organization" [99]

GT allows me to discover the state of DevOps from raw data without making presumptions. Therefore, my findings should cover all cultural, procedural, and technological aspects. This chapter discusses how I closely followed the GT methodology to perform an open-ended DevOps study.

3.1 Strauss and Corbin Grounded Theory Adoption

There are many varieties of GT practices. Researchers may choose the version of GT suitable for their research. I selected Strauss and Corbin Grounded Theory [89] for my DevOps study. Even within one version of GT, there are many optional processes. Researchers adopt the GT processes necessary to construct their theories. This section describes which Strauss and Corbin Grounded Theory processes, described in Section 2.3.2, I have adopted to construct my grounded theory in DevOps.

Following the Strauss and Corbin version of GT, in the first phase, I started by gathering raw data from field stakeholders. Then I used open coding to identify initial concepts, properties, and dimensions from the data. Through microanalysis, I organized the concepts into categories, and documented all findings with memos. Finally, I associated these categories with potential research questions. From these categories and potential research questions, I selected one topic for in-depth GT investigation. This topic has the potential to provide the highest value to stakeholders. From the categories and potential theories, I selected DevOps education as the topic for further investigation through elimination.

Following the Strauss and Corbin GT process, in the second phase, I identified different dimensions and perspectives of the selected topic, DevOps education, for in-depth examination and analysis. For each dimension and perspective, I selected the most appropriate GT process for further exploration. The GT processes I adopted included questioning, selective coding, surveying, data gathering, and making comparisons. I did not use a conditional-consequential matrix in analyzing different dimensions, since I did not find any appropriate situation to apply the mechanism. When analyzing different dimensions, I used memos to document my progress, thought, and findings in words and tables, which formed my final theory and report.

I did not finalize my theory using theoretical sampling, since my theory involved too many dimensions. It was not practical to find theoretical samples to encompass all the dimensions. Instead, I iterated the GT processes for each dimension until the theory was saturated.

3.2 Concrete Execution of Grounded Theory

Stol *et al.* found that most of the academic articles that claimed to be the result of GT projects did not actually employ GT [91]. Therefore, I decided to execute a concrete GT project that closely followed the GT processes. Throughout this dissertation, I will document how my project followed the GT processes step-by-step, and how I analyzed raw data to discover concepts and relationships. Others can use the documented steps to confirm the validity of this GT project.

Strauss and Corbin suggested that the investigators of a real GT project should start with open minds, and without a pre-determined agenda. Therefore, I followed this suggestion by not allowing my knowledge and preferences to affect my GT project. This section describes how I followed the GT processes to gather initial data without personal bias.

3.2.1 Initial Interviews

Through referral from the local IT practitioners, I invited IT practitioners interested in DevOps to participate in my initial raw data gathering interviews. I objectively summarized the six CI tasks [39] described in Section 2.1 and the maintenance tasks from "Continuous Maintenance" [34] for the interviewees. The six CI tasks are:

1. Continuous Code Integration
2. Continuous Database Integration
3. Continuous Testing
4. Continuous Inspection
5. Continuous Delivery
6. Continuous Feedback

The continuous maintenance tasks are:

1. Monitoring and reporting
2. Processes managing
3. Data cleanup, archive, and backup
4. Data analysis
5. Law compliance
6. Business continuous planning
7. Disaster recovery planning

Then I asked the interviewees questions regarding their DevOps experiences with repositories, build scripts, integrated development environments (IDEs), testing, code quality, and deployment. The questions were recorded in **Appendix A Interview Questions for Initial Raw Data Gathering**.

I interviewed five IT practitioners. In interview order, the interviewees had 6, 10, 30, 15, and 5 years of IT experience, with IT skills ranging from intermediate to advanced. The lengths of the interviews were approximately 26 minutes, 35 minutes, 31 minutes, 67 minutes, and 29 minutes. The interviewees provided IT services in different fields, with different responsibilities, but they had all participated in software development. As described in Section 2.1, DevOps grew from CI, which focused on software development, to include operations. Therefore, we only invited IT practitioners with software development experience. After the interviews, I transcribed 4 out of 5 interviews, since one interviewee preferred not to be recorded. Instead, I documented high-level information about the interview in a memo.

Following the GT processes, I applied open coding to each of the transcriptions. The next section will describe the open coding results in detail. Through open coding, I noticed that the volunteers provided consistent and similar ideas about DevOps. The GT methodology relies on quality, rather than quantity. Based on the coherent ideas among the interviewees, I found this diverse group of volunteers sufficiently covered enough of the DevOps spectrum. Therefore, I sought no further participants.

3.2.2 Open Coding

After I transcribed the interviews, another member of our research team and I followed the Strauss and Corbin GT open coding process to microanalyze the transcriptions individually. The purpose of open coding is to identify DevOps concepts and potential theories from the interviewees for further investigation. During the process, each of us went through each transcription line by line to code concepts, properties, and dimensions from the data. Codes were meaningful single-words or phrases that represented concepts. "A concept is a labeled phenomenon. It is an abstract representation of an event, object, or action/interaction that a researcher identifies as being significant in the data. … Example of concepts include a tornado, a flight, and a government agency, … they often provoke a common cultural imagery" (p.103) [89]. Through the microanalysis process, each of us linked concepts with thought, publications, and other studies, then documented the linkages in memos for future reference.

After coding the transcriptions individually, two of us got together to compare codes, line by line. We shared, compared, communicated, discussed, and enhanced the codes to agree on a final list recorded in **Appendix B Open Coding Result of the Initial Interviews**. During individual coding, each investigator selected words or phrases based on their understanding. Through discussion, we selected the most suitable words or phrases, which might or might not be the words used during the interview. If a concept appeared in the transcription multiple times, then the corresponding code would appear multiple times. After finalizing the codes of one transcription, we tried to apply the same codes to the rest of the transcriptions cumulatively and consistently. As a result, the same concept repeated in different transcriptions would have the same code.

Interviewee:

Really, continuous integration to me means you are getting it; your team is getting it.
It is visible everywhere.
And … ideally you can start with unit test.
But ideally, if you got other elements that's … long term test … stress test or scalability …
Whatever you have to do, you are working on that repo, those should be kicked off by Maven jobs.
You should be able to reverse as well.
Just giving unit tests … you may not … all that's grate …
I don't know.

Figure 3.1 A portion of an interviewee's response from transcription

For example, Figure 3.1 shows a portion of an interview transcription. A member of the research team and I went through the transcription individually to identify codes from the transcription. For the portion of transcription in Figure 3.1, the other member in the research team identified the code "different levels of CI", and I identified the code "progressive adaptation". When the two of us compared our codes, we agreed that we identified the same concept. After discussion, we decided to use the code "adaption level" to represent the concept.

I recorded the original codes from the transcriptions under the "Code" column in **Appendix B Open Coding Result of the Initial Interviews**. However, some of these codes were very specific, so I generalized them. For example, the code "Git", which referred to a specific product, was generalized as "tool". I recorded the generalized codes under the "Related Code" column. In addition, I recorded the related context of each code under the "Context/Expression from Transcription" column, if applicable.

3.2.3 Code Relationship Analysis

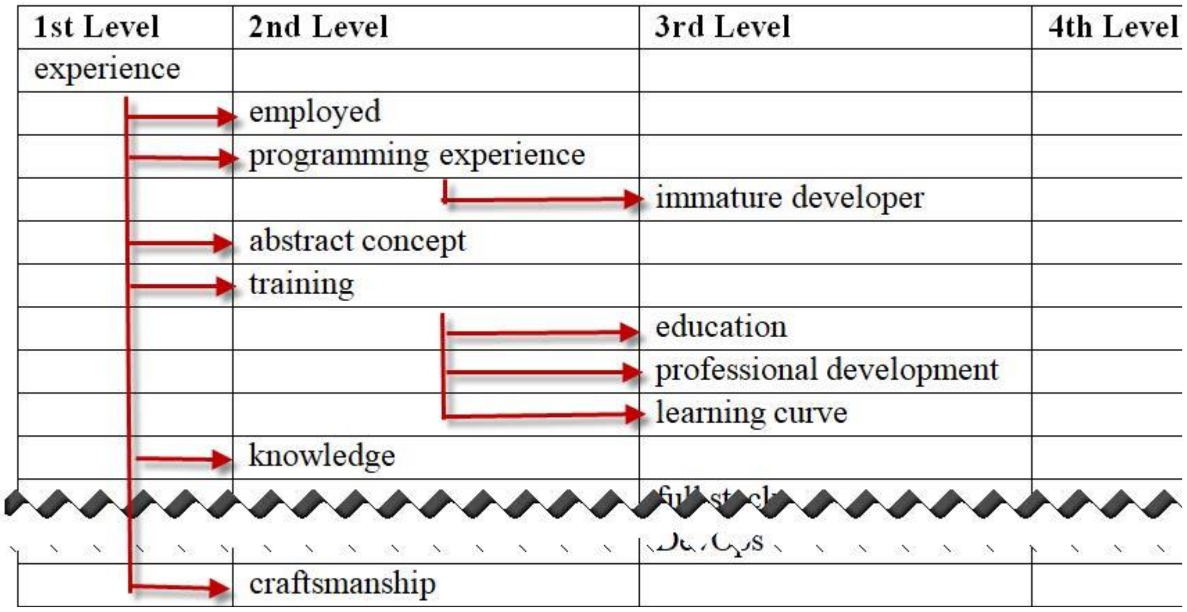
After the open coding and microanalysis processes, I finalized 34, 83, 28, and 125 codes from the four transcriptions respectively. There were 153 unique codes. Three of the 153 unique codes appeared most often: "experience" appeared 12 times, "automation" appeared 8 times, and "reduce defect" appeared 7 times. In addition, 90 codes appeared only once, 36 codes appeared twice, 17 codes appeared 3 times, 5 codes appeared 4 times, and 2 codes appeared 5 times. The codes covered a broad range of concepts across the DevOps spectrum. For example, the code "experience" was used in the context of programming experience, years of working experience, experience with a tool, experience in projects, and more. The next step in GT was to analyze the codes to identify important concepts raised across the interviews.

Code analysis started with identifying relationships between codes. I manually analyzed the codes to link related codes together. For example, the codes "education", "training", "experience", "knowledge", and "learning" were related according to their context listed in **Appendix B Open Coding Result of the Initial Interviews**. A code might relate to multiple other codes at the same time. For example, the code "knowledge" related to "learning", in terms of learning some knowledge. The code "knowledge" related to "skill", in terms of having knowledge about certain skill. The code "knowledge" also related to "tool", in terms of specific knowledge about a tool. The multi-dimension relationships between codes made the analysis difficult. The types of relationships (e.g. syntactic, example of, or generalized) are not significant in the first phase of the GT process. But it is important to identify all relationships between codes. Based on these relationships, investigator selects a potential topic for further GT study in the second phase.

Through code analysis, I grouped related codes into seven categories, named "experience", "practice", "benefit", "cost", "tool", "social aspect", and "third-party service". I organized the codes into category trees. The root nodes of the trees were the seven categories that represent the broadest concepts in the categories. The root node branched out to one or more child nodes for related codes. Each child node might further branch out to one or more child nodes. Each code might appear in multiple trees given the multi-dimensional nature.

For example, Figure 3.2 shows a portion of the code tree for the "experience" category. The root node of the tree is "experience", then "experience" branches out to child nodes "employed", "programming experience", "abstract concept", "training", "knowledge", and "craftsmanship". The node "training" further branches out to child nodes "education", "professional development", and "learning curve". The tree shows the relationships between codes related to "experience". The relationships in the tree are not definite, and the order of the nodes is not significant in GT.

The resulting trees are shown in **Appendix C Open Coding Trees**. The categories "experience", "practice", "benefit", "cost", "tool", "social aspect", and "third-party service" have 19, 50, 28, 14, 30, 18, and 6 unique codes respectively. The "third-party service" category has the least number of codes, and I excluded it from my further consideration. The "practice" category has the most number of codes (50), and the "tool" category has the second most number of codes (30). More codes in a category does not make the category more important. More codes reflects



1st Level	2nd Level	3rd Level	4th Level
experience			
	employed		
	programming experience		
		immature developer	
	abstract concept		
	training		
		education	
		professional development	
		learning curve	
	knowledge		
		full stack	
		DevOps	
	craftsmanship		

Figure 3.2 Portion of the "Experience" code tree

that stakeholders have more diversified interests in the category. As explained in Section 2.1, DevOps covers all IT processes from end to end, therefore the number of codes in the "practice" category is high. As described in Section 2.2.1.2, there are a large number of DevOps tools, therefore the number of codes in the "tool" category is also high. Code analysis identified these categories for possible further study: "experience", "practice", "benefit", "cost", "tool", and "social aspect".

3.2.4 Codes Occurrence Analysis

Among these codes and categories, I need to select a potential DevOps topic for further GT study. The selected topic should draw common interest among IT practitioners. To remain consistent with the original interviews, I looked for codes that appeared in multiple interviews. Table 3.1 lists the codes that appeared in three or more interviews.

Among all the codes only "cost", "reduce defect", and "training" appeared in all four interviews. These three DevOps topics had surely drawn attention from IT practitioners. In addition, there were 16 other codes that appeared in three out of four interviews. Only the code "feedback" appeared in more than one category, which were "practice", "benefit", and "cost". The categories "experience", "practice", "benefit", "cost", and "tool" have 3, 7, 5, 4, and 2 codes that appeared in three or more interviews respectively. The results showed that these codes represented common DevOps interest among IT practitioners.

EXPERIENCE		**PRACTICE**		**BENEFIT**		**COST**		**TOOL**	
Code	**#**	***Code***	**#**	***Code***	**#**	***Code***	**#**	***Code***	**#**
experience	3	auto build	3	feedback	3	cost	4	repository	3
knowledge	3	auto test	3	improve quality	3	feedback	3	tool	3
training	4	automation	3	maintainability	3	overhead	3		
		code integration	3	reduce defect	4	ROI	3		
		feedback	3	version control	3				
		merge	3						
		test data management	3						

Table 3.1 Categories of codes and their occurrences in number of interviews

The category "social aspect" had 18 codes, but none appeared in more than two interviews, therefore the "social aspect" category was not show in Table 3.1. The lower interest in the "social aspect" category agreed with the survey reported by DevOps Institute [12], which surveyed over 1600 IT practitioners regarding DevOps skill requirements. Accumulatively, the respondents rated "Soft Skills" less important than "Automation Skills", "Process Skills and Knowledge" [13].

From Table 3.1, the "practice" category has the highest number of codes, but "auto build", "auto test", and "automation" are closely related, which can be replaced by "automation". In this case, the five categories "experience", "practice", "benefit", "cost", and "tool" have almost the same amount of interests across the interviewees. Researchers may use the amount of interest to weight future DevOps research topics.

3.3 DevOps Grounded Theory Topic Selection

Through initial interviews, open coding, microanalysis, and code analysis detailed in Section 3.2, I identified potential DevOps topics for further GT study. From the topics, I identified potential research questions and theories. This section explains how I used elimination to select DevOps education for further study in my GT project.

3.3.1 Potential Research Questions Analysis

Throughout the coding and analysis processes, the research team recorded thoughts, ideas, concepts, and questions from interviewees in memos. After the code analysis, I reviewed the memos to document potential research questions.

In total, I documented 119 potential research questions listed in **Appendix D Potential DevOps Research Questions**. The research questions were grouped into the categories of "experience", "practice", "benefit", "cost", "tool", "social aspect", and "third-party service". There were 19, 38, 21, 5, 17, 17, and 2 questions in these categories respectively. The number of questions in each category correlated with the number of codes in that category, since each code represents specific ideas in the category. Scholars interested in DevOps may consider these 119 industrial-related questions in future research.

In the early days of this project, I mainly used the term CI, therefore the questions were mainly addressing CI. However, the questions covered CI, CD, and DevOps topics. As discussed in Section 2.1, most IT practitioners could not differentiate between CI, CD, and DevOps. To maintain the original sense of the research questions, I kept the term CI as they appeared in the interviews at the time.

3.3.2 Topic Analysis and Selection

Among the seven DevOps categories and 119 corresponding DevOps research questions, I needed to select one potential topic for further investigation. The rest of the GT project would focus on the selected topic. I reviewed the identified categories and questions, and looked for significant topics that represented common interests among IT practitioners. Broadly, the categories "experience", "practice", "benefit", "cost", and "tool" shared common interests among IT practitioners. After further analysis, Table 3.1 shows that the sub-categories "training" under "experience", "reduce defect" under "benefit", and "cost" (code) under "cost" (category) shared the most interests. Therefore, they were my top candidates for further GT investigation.

Since the category "social aspect" shared less interest among IT practitioners [13], I dropped it from my list of candidates. Future research should consider studying the "social aspect" of DevOps, as many IT practitioners believe that DevOps is a cultural movement.

Among the categories "experience", "practice", "benefit", "cost", and "tool", I considered which categories were more ready for further investigation at the time. During my first round of interviews, my interviewees showed diverse understandings and misunderstandings about DevOps practices. For example, one interviewee claimed to have adopted DevOps, only because they were using a version control repository. Another interviewee defined DevOps as "making frequent branching and merging in the repository". As shown in Section 2.1, DevOps practices are much more than using a version control repository and checking in source code frequently. IT practitioners seemed to lack a consistent definition for DevOps.

The 2015 DZone Guide to Continuous Delivery [56] reported that only 18% of the practitioners who were practicing CD performed the "textbook" implementation. Hence, most practitioners did not know or agree on what the DevOps practices were. The DZone Guide [56] also reported that 48% of the practitioners believed DevOps would become the IT standard in the future. Therefore, DevOps "practices" was immature for GT study, until after DevOps became more established with common practices emerged from the field.

Different DevOps practices lead to different costs and benefits. Hence, studies about "cost" and "benefit" must depend on studies about "practice", which was immature for study. Therefore, "cost" and "benefit" were not mature candidates for further investigation either. I also considered the "tool" category. As described in Section 2.2.1.2, DevOps employs many different tools, but none have become the de facto standard yet, so "tool" was also not a mature candidate. After this elimination, I decided to focus on the "experience" category.

In the "experience" category, the code "training" appeared in all of the interviews. Practitioners need appropriate training to learn DevOps. As one of the interviewees said, "Whatever tools you know actually guide your concept and how you will apply DevOps." Training strongly influenced what DevOps practices were adopted. Therefore, "training" was a mature candidate. As a result, I decided to study DevOps education further in my GT project. In the next section, I will describe how I defined a potential theory on DevOps education for further study in the second phase of my GT project.

3.3.3 Potential DevOps Education Theory

The goal of a GT project is to construct theory. After I selected the topic, I defined a potential theory on the topic, incorporating information from the microanalysis. Chapter 4 will evaluate the potential theory based different dimensions and perspectives. During the evaluation process, the theory may be enforced, enhanced, refined, rejected, or redefined. At the end of Chapter 4, a DevOps theory will emerge from the analysis.

To define a potential theory regarding DevOps education, I reviewed the original transcriptions, the memos, the code analysis, and the research questions. From the transcriptions, I realized that all of the interviewees acquired their DevOps skills mainly from employment. As one of the interviewees said, "DevOps is an experience-based movement." In **Appendix C Open**

Coding Trees, the code "training", appears in Table C.1, under the category "experience". The code "training" has the sub-categories: "professional development", "learning curve", and "education". The relationship between these codes indicates that IT practitioners actively acquired their DevOps skills through hands-on working experience, instead of passive training. Since I decided to focus on DevOps education, I was interested to study whether IT practitioners acquired DevOps skills through formal training or through hands-on working experience.

In addition, I reviewed the 119 potential research questions listed in **Appendix D Potential DevOps Research Questions** for DevOps education related questions. I found the following questions in the "Experience" category to be relevant. The potential theory could possibly answer some of these questions.

- Is there a need to include CI education in computer science programs in institutions?
  (Investigate whether CI can be taught in academia if needed.)
- Does CI depend heavily on individual's experience?
  (Investigate how CI practitioners learn CI from experience.)
- What are the skills and knowledge needed for CI?
  (Investigate what CI practitioners need to learn.)
- Do existing education curricula cover CI?
  (Investigate whether CI practitioners can learn CI from academia.)
- How CI specialists get their training?
  (Investigate how CI practitioners learn their skills.)
- What are the requirements for CI jobs?
  (Investigate what CI practitioners are required to learn.)
- How popular are CI positions in job posting sites?
  (Investigate the professional value in learning CI.)
- What should be the qualification of a CI specialist?
  (Investigate the learning goals set for CI qualification.)
- How do IT practitioners learn CI?
  Are there differences between junior and senior IT practitioners?
  (Investigate how different levels of CI practitioners learn their skills.)
- Is CI pure craftsmanship?
  (Investigate whether CI should be learned as craftsmanship, instead of skills.)

- Do CI experts need to be "Jacks-of-All-Trades"?
  (Investigate whether CI requires a specific skillset, or undefined skillset.)

Based on these questions, I wanted to define a potential theory that contrasts learning DevOps through formal training against hands-on experience. Strauss and Corbin suggested that a potential theory should be abstract and catchy to invite deeper thought. The potential research question "Is CI pure craftsmanship?" stood out as captivating. I traced the question back to the code "craftsmanship". The code was identified during open coding to describe a situation where the interviewee adopted a container into their DevOps pipeline through trail-and-error. The process was like creating a craft, a piece of art, from their mind by experience. There was no training or help available. Therefore, I named my potential theory "Continuous Integration (CI) is a craftsmanship. How do you learn the craft?" I hoped the conceptual theory would invite profound reflection. The full text of the potential theory is provided in **Appendix E Potential Grounded Theory**. When I defined the potential theory, I used the term CI, instead of DevOps, therefore I left the term as-is. Through this potential theory, I would compare learning DevOps in academia versus industry.

In conclusion, I started my GT project by interviewing IT practitioners about DevOps. Through open coding and code analysis, I identified seven categories of DevOps topics that interested the IT practitioners. The categories were: "experience", "practice", "benefit", "cost", "tool", "social aspect", and "third-party service". Through elimination, I decided to further investigate DevOps education, in the category of "experience", and I defined a corresponding potential theory "Continuous Integration (CI) is a craftsmanship. How do you learn the craft?" Chapter 4 describes how I studied DevOps education through evaluating this potential theory based on different dimensions and perspectives.

4 Grounded Theory Study about DevOps Education

Chapter 3 describes how I selected DevOps Education for further GT investigation, and defined the potential theory "Continuous Integration (CI) is a craftsmanship. How do you learn the craft?" In this chapter, through in-depth GT study, I answer the following questions: how do people learn DevOps? Is DevOps education available in academia and/or in industry? What kind of DevOps education is available? I studied these DevOps education questions from academic and industrial perspectives, through questioning, selective coding, surveying, data gathering, and making comparisons.

I first investigated the academic perspective. In Section 4.1, I investigated whether academic education conveyed DevOps knowledge to students. In Section 4.2, I investigated whether academics taught DevOps. In Section 4.3, I investigated whether academic research would adopt DevOps. Then I investigated the industrial perspective. In Section 4.4, I investigated the job requirements of DevOps practitioners. In Section 4.5, I investigated what kind of DevOps professional training was available. In Section 4.6, I investigated how the experts learned their DevOps skills. Through these investigations, I identified the state and challenges in DevOps education and looked for ways to improve future DevOps education.

4.1 DevOps in Academic Education

According to the ACM & IEEE Guidelines for Undergraduate Degree Programs in Computer Science, "The education that undergraduates in computer science receive must adequately prepare them for the workforce" [100]. DevOps has become a requirement for the workforce, but the 2015 DZone Guide to Continuous Delivery [56] reported that most IT practitioners lacked DevOps knowledge, even though most IT practitioners received formal education. Did IT practitioners learn DevOps during their academic education? To answer this question, I searched computer science curricula in academic institutions to learn whether academic education delivered DevOps knowledge. The data showed that computer science curricula covered limited DevOps knowledge.

4.1.1 Academic Education Search Process

I searched computer science curricula for DevOps content to determine whether DevOps knowledge was conveyed in undergraduate education. However, there were many institutions in the world; I decided to sample the top 50 institutions listed in the 2017 QS World University Rankings by Subject - Engineering and Technology [101]. Since these were the top technology institutions in the world, their computer science curricula should be representative. The list of institutions is included in **Appendix F 2017 QS World University Rankings by Subject - Engineering and Technology**.

For each institution, I searched the institutions' home page for the computer science department, then looked for undergraduate computer science courses offered by the department. In the course description and curriculum, I looked for DevOps related terminology to determine whether the institution conveyed DevOps knowledge to its students. To determine what DevOps related terminology to look for, I reviewed these six fundamental CI tasks defined by Duvall *et al.* [39]:

1. Continuous Code Integration
2. Continuous Database Integration
3. Continuous Testing
4. Continuous Inspection
5. Continuous Delivery
6. Continuous Feedback

Then, I compared these six tasks with the codes from the initial interviews in **Appendix B Open Coding Result of the Initial Interviews**, since the codes represented what IT practitioners understood as DevOps processes. Among the overlapping terminology, I identified five groups of significant DevOps terms for the curriculum search. They are:

- **Continuous Integration** (continuous, integration, delivery)
- **Testing** (**testing**, test)
- **Build** (auto, **build**)
- **Repository** (**repository**, version control, git, cvs)
- **Deploymen**t (deploy, **deployment**, release)

Variations of the terminologies were listed in parentheses. I searched for each variation in course descriptions and curriculum. For each match I reviewed the context to make sure the content actually referring to the corresponding DevOps topics.

4.1.2 Academic Education Search Result

The first challenge of the process was finding the computer science department of each institution. First, the computer science departments had different names depending on the languages and cultures of the institutions. In most institutions, computer science was an individual department under engineering, science, or mathematics using different names. In some institutions, computer science was a school by itself. Moreover, in some other institutions, computer science undergraduate programs were offered by departments with different names, for example electrical engineering. In this situation, I identified the smallest department in the institution that offered computer science (software) related education as the target of my search.

I excluded 11 institutions from the 50 because they have no English course descriptions, or their course descriptions were only accessible by authenticated users. Some institutions provided course outlines only, other provided detailed descriptions or curriculum. I always explored the most detailed description available. For each DevOps term found in the description, I reviewed the context to make sure the course actually included DevOps related content. The only exception was on "deployment". Some course descriptions included the term, but provided no context. I included those courses in my result, but marked by asterisk (*). Table 4.1 shows my search result.

Rank	Institution	CI	Test	Build	Repo	De-ploy
1	Massachusetts Institute of Technology (MIT)	0	4	0	0	0
2	Stanford University	0	0	0	0	1*
3	University of Cambridge	1	3	0	1	0
4	Nanyang Technological University (NTU)	0	2	0	0	1*
5	ETH Zurich - Swiss Federal Institute of Technology	0	2	0	0	0
6	Imperial College London	2	4	1	0	1
7	National University of Singapore (NUS)	0	10	1	2	1
8	University of California, Berkeley (UCB)	0	1	0	0	0
9	University of Oxford	0	1	0	0	0
10	Tsinghua University	--	--	--	--	--
11	The University of Tokyo	--	--	--	--	--
12	Ecole Polytechnique Fédérale de Lausanne (EPFL)	0	1	0	0	0

Rank	Institution	CI	Test	Build	Repo	De-ploy
13	Harvard University	0	2	0	0	0
14	Korea Advanced Institute of Science & Technology (KAIST)	0	1	1	0	0
15	The Hong Kong University of Science and Technology (HKUST)	0	3	0	0	0
16	Georgia Institute of Technology	0	3	1	1	0
17	California Institute of Technology (Caltech)	0	4	0	1	0
=18	Peking University	--	--	--	--	--
=18	Tokyo Institute of Technology	0	0	0	0	0
20	Delft University of Technology	0	2	0	0	0
21	Seoul National University	0	0	0	0	0
22	Kyoto University	--	--	--	--	--
23	National Taiwan University (NTU)	--	--	--	--	--
=24	Politecnico di Milano	0	1	0	0	0
=24	Shanghai Jiao Tong University	--	--	--	--	--
=24	Technical University of Munich	--	--	--	--	--
27	The University of Hong Kong	0	2	0	0	0
28	The University of Melbourne	0	3	0	0	0
29	KTH Royal Institute of Technology	1	5	1	1	0
30	University of California, Los Angeles (UCLA)	0	3	0	0	0
=31	Carnegie Mellon University	0	0	0	0	0
=31	The University of New South Wales (UNSW Sydney)	0	7	0	0	0
33	RWTH Aachen University	--	--	--	--	--
34	University of Toronto	0	3	0	1	1*
=35	Technische Universität Berlin (TU Berlin)	--	--	--	--	--
=35	University Malaya (UM)	0	5	0	0	0
37	Princeton University	0	1	0	0	0
=38	KIT, Karlsruhe Institute of Technology	--	--	--	--	--
=38	National Tsing Hua University	0	1	0	0	0
=38	University of Illinois at Urbana-Champaign	0	4	0	0	0
41	The University of Sydney	0	5	0	1	1*
42	Zhejiang University	0	1	0	0	0
43	Technical University of Denmark	1	8	0	2	1
=44	Monash University	3	8	0	2	2
=44	University of Texas at Austin	0	2	0	0	0
46	The Australian National University	0	4	0	1	1*
=47	Cornell University	0	4	0	1	0
=47	Fudan University	--	--	--	--	--
49	University of Michigan	0	3	0	0	0

Rank	Institution	CI	Test	Build	Repo	De-ploy
50	The Chinese University of Hong Kong (CUHK)	0	3	0	0	0
	Total Number of Courses	8	116	5	14	10
	Total Number of institutions	5	35	5	11	9

* Description included the searched term, but not sure whether the content was CI related.

Table 4.1 Number of Courses Including CI Content in Institutions

4.1.3 Academic Education Search Analysis

From the search result, only 5 out of 39 (12.8%) institutions taught about CI; 5 out of 39 (12.8%) taught about building application; 11 out of 39 (28.2%) taught about repository; 9 out of 39 (23.1%) taught about deployment. However, excluding the 5 courses, with questionable deployment content, only 4 out of 39 (10.3%) institutions taught about deployment. Nevertheless, 35 out of 39 (89.7%) institutions taught about software testing. Hence, DevOps and its corresponding tasks were not part of the curriculum in most institutions, except software testing.

4.1.3.1 *Software Testing Teaching*

Almost all institutions taught software testing. One possible explanation is that software testing has always been an important component of software engineering. Myers published the book "The Art of Software Testing" [102] in 1979. Myers found that "students graduate and move into industry without any substantial knowledge of how to go about testing a program" [102], therefore he wrote the book to satisfy the need. In that case, DevOps may be added to the computer science curriculum in the future, when DevOps becomes essential to students going into industry.

Software testing continues to be an important component of software engineering. In 1999, Back introduced Test-Driven Development (TDD), as part of Extreme Programming (XP) [40]. TDD emphasizes that developers should program test cases before programming functionalities. In 2007, Duvall *et al.* [39] defined test automation as a requirement of CI, hence DevOps. However, by most of the course description, I could not tell whether the courses included TDD and automated testing. If I looked strictly for automated software testing during my course description search, then the number of resulting courses would be much lower.

I also realized that most of the courses identified in Table 4.1 were advanced software engineering courses, but none were dedicated to DevOps. Hence, DevOps was only a topic in a larger software engineering course. In the case of software testing, only six institutions, Korea Advanced Institute of Science & Technology (#14), California Institute of Technology (#17), Delft University of Technology (#20), University Malaya (#35), University of Illinois at Urbana-Champaign (#38), and Monash University (#44), have dedicated courses to teach automated testing. In addition, The University of Sydney (#41) and Technical University of Denmark (#43) have dedicated courses about version control and testing. Academic education has embraced testing, one of the DevOps topics, but not yet DevOps.

4.1.3.2 ***Challenges in DevOps Teaching***

During the search, I also realized that it might not be easy to teach some of the DevOps skills in an academic setting. For example, to let students practice automated software build and deployment, students needed designated build and deployment infrastructure. Students might also need special authentication and local area network zoning to avoid conflicts. Even with the provision of virtual machines, it might be a challenge to configure infrastructure for many students in an academic environment.

Nonetheless, the course "COMP1040 The Craft of Computing" offered by The Australian National University (#46) agreed with my potential theory that software development was craftsmanship, a form of art. "The focus of The Craft of Computing will be on understanding core principles that allow students to quickly and confidently learn and apply a variety of computational tools to several different types of problem" [103]. As the course description indicated, academic education could only equip students with the fundamental knowledge. Students needed to advance their comprehension and understanding through hands-on experience.

Academic education also had difficulty catching up with the fast growing industrial practice. As the Department of Computer Science at the University of Texas at Austin (#45) indicated, "The [Computer Science] faculty revised the curriculum, because computer science has become too large to learn in four years" [104]. Computer science research progresses and day-to-day applications are expanding rapidly in academia and everyday lives. Institutions are challenged to define curricula that would properly equip students to be IT practitioners. DevOps covers all IT processes from end to end. To provide comprehensive DevOps education, institutions need to add much more content to the already packed four-year curriculum. Therefore, academic education can cover, at most, high-level DevOps knowledge.

With the ever-expanding amount of computer science knowledge, some institutions chose to focus more on DevOps. From Table 4.1, institutions offered on average 3.92 DevOps related courses and the median was 3 courses. However, the National University of Singapore (#7) offered 14; the Technical University of Denmark (#43) offered 12; and the Monash University (#44) offered 15. On the other hand, many more institutions did not address DevOps. Three institutions, Tokyo Institute of Technology (#18), Seoul National University (#21), and The University of New South Wales (#31), did not offer any DevOps related courses. Stanford University (#2) has offered only one course that might include deployment content. Seven other institutions offered only one course about testing. Different institutions have different favourite areas of teaching.

After reviewing the curricula of the top 39 institutions listed in the 2017 QS World University Rankings by Subject - Engineering and Technology, I found that most institutions were not conveying sufficient DevOps content. With DevOps becoming an industrial standard, all institutions should start conveying basic DevOps knowledge to their students, such as the six fundamental CI tasks defined by Duvall *et al.* [39]. Since institutions may update what they teach without updating their curriculum, the curriculum may not reflect what is actually being taught. In the next section, I describe a survey of academic educators about whether they teach DevOps content in their undergraduate programs.

4.2 **DevOps in Academia**

From my curriculum search, I learned that most institutions had yet to include DevOps in their undergraduate curriculum. It was possible that the academics were actually teaching DevOps, but they have not updated the curriculum to include DevOps terminologies. Therefore, I decided to survey the academics directly to explore the DevOps education condition in institutions. Unfortunately, the DevOps survey failed to draw participation from the academics.

4.2.1 Academic Survey Process

Academics are busy people, so I designed a short survey to discover the DevOps teaching condition in institutions. The survey should take less than one minute to answer, in the hope of receiving numerous responses. Since I have searched the curricula of the top 50 institutions in the 2017 QS World University Rankings by Subject - Engineering and Technology [101], it would be consistent to survey the computer science academics in these institutions. In addition to the computer science academics, I also invited the electrical engineering academics to participate in the survey as comparison, since I found during my curriculum review that many institutions grouped the electrical engineering department with the computer science department together. I asked the academics these six questions:

1. Are you an engineering or computer science educator?
2. How many years have you been teaching undergraduate engineering or computer science courses?
3. How many years of full-time hands-on experience as an engineer or IT practitioner do you have?
4. Do you teach students about automated unit testing in your undergraduate courses?
5. Do you teach students about Continuous Integration (CI) in your undergraduate courses?
6. Do you teach students about DevOps in your undergraduate courses?

Question 1 and 2 collected background information about the respondents. Question 3 found out whether the respondents have hands-on working experience. Hands-on working experience is very important to DevOps, since IT practitioners claimed that "it [took] many years of experience, combined with a solid understanding of tools, to eventually become a truly effective Senior DevOps practitioner" [105]. My potential theory also projected that DevOps education substantially depended on hands-on experience. Knowing whether the respondents had hands-on working experience helped me understand what kind of DevOps knowledge the respondents might possess. Questions 4 to 6 collected data about whether the respondents were teaching automated testing, CI, and DevOps in undergraduate education. The survey result would clarify whether academics were teaching DevOps.

Therefore, I searched the websites of the computer science and electrical engineering departments of the top 50 institutions again to extract contact information of their department heads. Through the contacts, I invited the academics in the computer science and electrical engineering departments to participate in my survey.

4.2.2 Academic Survey Result and Analysis

Among the top 50 institutions, I excluded three institutions from my survey invitation. Those three institutions did not have English contact information. I sent my survey invitation to the computer science and electrical engineering department heads. I asked the heads to forward the invitation to all educators in the departments. A sample of the invitation letter was included in **Appendix G Academic Survey Invitation Letter**.

The survey was published as a Google Form at https://docs.google.com/forms/d/e/1FAIpQLScZ1eSW9eOUHdTuq3ixOnTBfptZSx4jzqN5FoR8B3LnMt0IMA/viewform on Feb 1, 2018 for two months. The survey is included in **Appendix H Survey comparing hands-on experience between Engineering and Computer Science educators**. Unfortunately, after two months, only four responses were received, one from computer science and three from engineering. The number of responses was insignificant for any conclusion. The only observation was that the single computer science respondent did not teach automated testing, CI, nor DevOps, but one of the three engineering respondents taught automated testing, CI, and DevOps. Hence, DevOps is not only a computer science topic; it is also an engineering topic.

With such a low number of responses, my survey result was inconclusive. I did not know whether academics were not interested in DevOps, or were too busy to answer the survey. However, the lack of academic participation was consistent with my finding in Section 2.2.2 that the academic publications in DevOps were relatively low. Since I could not draw academics to my survey, I talked to some academics directly regarding their DevOps knowledge, experience, and expectations, as described in the next section.

4.3 DevOps in Academic Research

My potential theory proposes that DevOps is craftsmanship, which requires hands-on experience and practice to refine. I tried to study whether academics practice DevOps in research projects. My results showed that academics were not likely to practice DevOps, for the cost of adopting DevOps was higher than its benefit. Since academics did not want to practice DevOps, it was unlikely that they have the knowledge or interest to teach students about DevOps, which supported my findings in Section 4.1 and 4.2.

4.3.1 Graduate Student Interview Process

Graduate students usually implement computer science research projects, under the supervision of computer science professors. Therefore, I was interested to learn whether graduate students practiced DevOps in their research projects. I invited all graduate students in my local computer science department for interviews by sending them the invitation in **Appendix I Graduate**

Student Research Invitation Letter. In addition, I personally talked to the administrators of different research groups to invite graduate students from different research areas for interviews.

I learned in Section 4.1 that DevOps was not a popular topic in academic education, and graduate students might not know about DevOps. Therefore, I started each interview with a 10-minute DevOps presentation. The slides of the presentation are included in **Appendix J Continuous Integration (CI) Introduction for Graduate Students**. After the brief presentation, I asked the graduate students the following questions.

1. Does your research involve any coding?
2. Do you think adopting CI in your situation will improve the quality of your research?
3. Do you think you would adopt CI in your situation? Why and why not?
4. What is the biggest barrier in adopting CI? Emotional? Intellectual? Technical?

Just as the initial interviews described in Chapter 3, I transcribed the graduate students' interviews. Unlike the initial interviews, I applied selective coding, instead of open coding to the transcriptions. Through selective coding, I went through the transcriptions looking for data to verify or reject the potential theory. The potential theory would be refined through the process.

4.3.2 Graduate Student Interview Result

After a month long effort, all research groups turned down my invitation. As one of the groups responded, "nobody [in the group] seem[ed] to be interested in learning more about [DevOps]". The rejection was itself very valuable data for my GT project. It was clear that academics were not interested in DevOps. In the end, I interviewed three graduate students in different research groups, who have IT industry experience in different fields. All three interviewees provided consistent answers to my questions. Therefore, I determined that there was sufficient data to saturate my theory and seek no addition interviewees.

For the first question, all interviewees developed software for their research projects. For the second question, after receiving the DevOps presentation, all interviewees agreed that DevOps would improve the quality of their software projects. For the third question, all interviewees, without any doubt, were sure that they would not adopt CI in their research projects. They provided the following reasons for not adopting DevOps.

- Researchers lacked knowledge and experience about DevOps.
- DevOps sounded really complicated.
- The DevOps learning curve was too steep.

- It took a lot of time and trial-and-error to set up a DevOps environment.
- DevOps tools were not user friendly and lacked documentation.
- Research support teams did not have sufficient DevOps resources and support.
- Research projects were too small and short for DevOps.
- The overhead of setting up DevOps did not fit the tight timeline of research projects.
- DevOps did not add enough value to research projects.
- DevOps was only for commercial software projects.

4.3.3 Graduate Student Interview Analysis

The rejections and the low number of volunteers provided evidence for the lack of interest among academics. When I approached the graduate students who had not worked in the IT industry, they either had never heard about DevOps, or had no interest in learning anything about it. All three of the volunteer interviewees had working experience in the IT industry. Hence, DevOps drew interest from people with hands-on working experience only, as I had claimed in my potential theory. This phenomenon may hinder DevOps education, since undergraduate students are not likely to have IT working experience.

It was interesting that all three interviewees firmly showed that they would not adopt DevOps in their research projects, even though DevOps would benefit their projects. One of the barriers was that the cost of adopting DevOps was bigger than the benefit. In deeper discussion with the interviewees, replicability was the biggest benefit of DevOps. By adopting DevOps, research results would be easily repeatable and verifiable by the public, which improved replicability and credibility of the result. However, higher replicability was not enough to persuade the interviewees to adopt DevOps. As summarized in one of the interviews: "with [DevOps], I can make research result repeatable, but until it is commonly adopted by the whole research field, no one wants to spend the time to try to make it happen". If academia gives more credit to verifiable results, then there may be more interest in DevOps.

My interview identified barriers to adopt DevOps in academia. A survey of 497 IT practitioners found similar barriers in industry, as quoted below.

Top four barriers in CD adoption were:

1. *Corporate culture - not enough collaboration or DevOps practices (53%)*
2. *Lack of time (47%)*

3. *Engineers or operation team do not have the right skill set (43%)*
4. *No support from management (30%)*

Top four barriers in CD implementation were:

1. *Environment configuration and setup (56%)*
2. *Coordination of team member and resources (34%)*
3. *Regression testing (32%)*
4. *User acceptance testing (31%)*

- The 2017 DZone Guide to DevOps Volume IV [58]

In both academic and industrial environment, participants were concerned about lacking knowledge, lacking skills and resources, difficulties in setting up the environment, and constrained timelines. Since academics are not motivated to learn or practice DevOps, it is not likely that they have the knowledge or interest to teach DevOps, which is consistent with my findings in the previous sections. Much works can be done to improve DevOps recognition in academia. For example, some of the interviewees mentioned that their participation in the DevOps surveys and interviews increased their knowledge and interest about the topic. Therefore, research on the topic would raise DevOps recognition in academia. Starting in the next section, I describe my in-depth study of DevOps education from the industrial perspective.

4.4 DevOps Requirements in Industry

Both academics and industrial practitioners believed that lacking of DevOps knowledge and skills was a barrier. Therefore, I identified the required DevOps knowledge and skills. Knowing what the required DevOps skills are, will clear the path for DevOps education. If DevOps is craftsmanship that required hands-on experience, then IT practitioners need to know what they should put their hands on. Unfortunately, my study showed that DevOps lacks a clear requirements specification. IT practitioners need to comprehend the whole spectrum of IT processes through hands-on experience, so they can apply DevOps knowledge and skills to connect all IT processes together.

4.4.1 DevOps Jobs Requirements

Since DevOps is very popular in industry, IT practitioners wonder what the required DevOps knowledge and skills are. Lee searched through DevOps engineering job postings and identified eight groups of required skillsets corresponding to the eight groups of tasks, which I summarize as [106]:

- Design, build, and operate technology stack.
- Configure, monitor, and manage systems.
- Program different tools in various languages.
- Script Linux/Unix processes with various languages.
- Operate different cloud products.
- Automate IT processes regardless of tools.
- Understand and enforce best practices.
- Being proficient in interpersonal communication.

This long list of requirements cover almost the whole spectrum of IT processes and tools. DevOps practitioners need to have the knowledge and skills of architect, programmer, scripter, system administrator, database administrator, cloud administrator, operational analyst, and more. They also need soft skills, including management, leadership, and communication. In conclusion, there are no clearly defined requirements for DevOps engineers. The more experience the better. As Lee concluded from the job postings in LinkedIn [107], DevOps engineers require at least five years of IT practitioners experience [106]. Hence, IT practitioners cannot become DevOps practitioners simply through education, without hands-on working experience. However, IT practitioners could learn fundamental DevOps knowledge from academic education, and advance their DevOps skills at work.

Delanbanque, a DevOps practitioner recruiting specialist, also agreed with Lee that DevOps job postings "change[d] entirely from company to company" [108]. Delanbanque suggested hiring "someone who under[stood] the methodologies over the tools" [108] for DevOps positions. As my potential theory suggested, DevOps is a form of art. Craftsman use different tools to create art pieces they comprehend through hands-on experience. "The focus should be on the candidates’ ability to solve problems" [108]. Therefore, hands-on working experience is compulsory for DevOps practitioners.

4.4.2 DevOps Job Search Process and Results

According to the responsibilities identified by Lee [106], the requirements for DevOps practitioners are broad and extensive, so I studied whether DevOps positions actually exist. I searched three types of job sites for DevOps related jobs: a general public job site, an IT practitioner specific job site, and an academic specific job site. Using these three types of job sites, I confirmed that DevOps jobs actually exist, and they are popular.

For a general public job site, I selected Monster.com [109] for the United States. Monster is a global online employment service provider, and I chose the United States to limit the search results. For an IT practitioner specific job site, I selected DZone.com. DZone [110] is one of the world's largest online communities, and is a leading publisher of information and resources for software developers. For an academic specific job site, I selected IEEE Computer Society Jobs Board [111]. IEEE Computer Society is one of the primary academic research publishers. IEEE Computer Society receives a lot of attention from academic researchers and IT professionals. Since the three job sites have different data and styles, I used different search criteria for each site.

Monster's postings are very broad and the number of jobs is very large, which increased the difficulty of my search. A simple search by keyword results in more than 1000 postings. Monster shows "1000+ Jobs Found", but not the actual number of matching postings. Therefore, I had to limit my search using extra criteria. When I performed my search on December 13, 2017, I further limited my search for postings that were posted in the last seven days and located in California. In addition to the posting date and location, I further limited my search by job titles. Monster posts its popular job titles in https://www.monster.com/jobs/job-title. From the list, I looked for titles with the keywords, "continuous", "integration", "devops", and "software". The Monster's job titles with the corresponding keywords are listed in Table 4.2.

Job titles can be very confusing. Therefore, for each job title in Table 4.2, I reviewed the first 20 postings from the search results to determine whether any of the 20 postings were actually DevOps related. If not, the titles were excluded from the final search list. The job titles with DevOps related postings are marked by an asterisk (*) in Table 4.2. I searched for the number of posts available with those titles in the last 7 days in California on December 13, 2017. For comparison, I also searched for three commonly used software developer titles: Software Analyst, Software Architect, and Software Engineer. Table 4.3 shows the search results.

Keyword	Titles
continuous	Continuous Improvement, Continuous Improvement Analyst, Continuous Improvement Consultant, Continuous Improvement Coordinator, Continuous Improvement Engineer, Continuous Improvement Facilitator, Continuous Improvement Lead, Continuous Improvement Manager, Continuous Improvement Officer
integration	Application Integration Architect*, Application Integration Engineer*, CRM Integration Specialist, Director Systems Integration*, Market Integration Manager, Network Integration Engineering, Software Integration Engineer*, System Integration Analyst*, System Integration Architect*, System Integration Consultant*
devops	DevOps Engineer*
software	Software Analyst, Software Architect, Software Engineer, Software Engineering, Software Engineering Manager, Software Instructor, Software Integration Engineering*, Software Quality Assurance, Software Quality Assurance Coordinator, Software Quality Assurance Engineer

* Job titles with CI/CD & DevOps related postings.

Table 4.2 Job Titles and Number of Postings in Monster with the Specific Keywords

CI/CD & DevOps Related Job Title	# Post	Other Job Title for Comparison	# Post
DevOps Engineer	489		
Application Integration Engineer	1000+		
Director Systems Integration	53		
Software Integration Engineer	516		
System Integration Analyst	35	Software Analyst	969
System Integration Architect	17	Software Architect	173
System Integration Consultant	1000+	Software Engineer	1000+

Table 4.3 Number of Postings by Job Title in Monster

DZone is a community for software developers. Its members are experienced IT practitioners. Therefore, its postings have an IT focus. DZone classifies job postings by the categories in Table 4.4. Employers can select any number of these categories for their postings. On December 13, 2017, I searched DZone for postings using the categories "DevOps" and "Integration". On that date, DZone had about 100 postings. The "DevOps" category had 15 postings; seven with "DevOps" in the job titles, eight without. The "Integration" category had seven postings; one with "Integration" in the job titles, six without.

IEEE Computer Society's main audience is academic researchers and IT professionals. Therefore, its postings target academics, graduate students, and IT professionals. There were not many job postings in the IEEE Computer Society Jobs Board compared with Monster. I searched on December 14, 2017 for job postings with titles or description including the keywords

Agile	DevOps	IT Management	Software Architecture
Back-End	Engineering	Java	System Engineering
Big Data	Front-End	Microservies	Support
Cloud	Full Stack	Mobile Dev	UX/UI
Data Science	Integration	Performance	Web Development
Database Admin	Internet of Things	Security	

Table 4.4 DZone Job Categories

"DevOps", "Continuous", and "Integration". On that date, the IEEE Computer Society Jobs Board had 336 postings. Searching by the keyword "DevOps" returned one posting. Searching by the keyword "Continuous" returned 90 postings. Searching by the keyword "Integration" returned 62 postings.

4.4.3 DevOps Jobs Search Analysis

Based on my search results, DevOps has definitely become a term used in job titles. I found DevOps as part of job titles in all three types of job sites. However, there was no job title that used the terms CI or CD. As I have described in Section 2.1, most IT practitioners considered CI, CD, and DevOps as one thing.

On Monster, there were 489 postings with the title DevOps Engineer, and 969 postings with the title Software Analyst. Software analyst is a common job title for software developers and programmers. In my search, DevOps Engineer had half the number of postings of Software Analyst. Hence, DevOps Engineer has become a common title among IT practitioners.

On Monster, there were many more postings for Software Analyst (969) than Software Architect (173). Hence, the demand for Software Analysts is much higher than the demand for Software Architects. The number of postings for each title represents the corresponding demand in industry. Similarly, the number of postings for DevOps Engineer (489) was about half of those for Software Analyst (969) and almost three times of those for Software Architect (173). This represents a significant demand for DevOps practitioners. In addition, in DZone, 15 out of 100 postings were categorized as DevOps. Hence, DevOps represents a significant portion of IT positions among the IT practitioners, especially in the software development community.

Even though there were many postings for DevOps Engineer in Monster, sampling the job descriptions showed that most of the postings had either a development focus or an operations focus. Delanbanque had the same findings that employers were "looking for a candidate from either operations and infrastructure background or someone from a software engineering and development background" [108]. This condition indicates that the DevOps movement is still far from its original goal to integrate development with operations. However, to fulfill the original DevOps goal, IT practitioners need to have extensive knowledge and skills, as identified by Lee [106] in Section 4.4.1. It is not likely an academic education can cover the required extensive knowledge and skills. IT practitioners need to earn and enhance their DevOps knowledge and skills through hands-on experience.

Since DZone is a software developer community, the job requirements for DevOps practitioners in DZone's postings are much closer to those identified by Lee [106]. This condition represents the challenges of the human resource representatives trying to hire DevOps practitioners. Only experienced IT practitioners understand what DevOps really stands for, and what kind of work experience is significant for DevOps. Human resource representatives without IT experience may not be able to identify the DevOps knowledge and skills from applicants' resumes. If the IT industry does not know what skills they want to employ, then it is hard to design DevOps education. As the executive director at Robert Half Technology said, "At some point in the future you will be able to hire for just DevOps, [but] we're almost ahead of ourselves today. What you really need or want is still somewhat undefined." [112]

Finally, in Monster, many jobs with "continuous" and "integration" in their titles had nothing to do with DevOps. It was an indicator that the significance of CI, CD, and DevOps was not well established in the general public job site. It will take more time for DevOps to be mature and recognized by the broader IT audiences.

After reviewing job postings in three different types of job sites, it is clear that DevOps practitioners are highly demanded in the IT industry. However, the role and identity of DevOps practitioners is not well established. In the next section, I describe the industry's effort in establishing DevOps professionalism.

4.5 DevOps Professionalism

Authoritative entities or formally recognized training programs could lead to the maturity of DevOps. Therefore, I studied whether authorities or training programs exist for DevOps. Many organizations are trying to establish authority, guideline, and standard for DevOps through writing, community building, conferences, and training programs. To identify DevOps authorities, I searched for DevOps certifications, training programs, communities, events, and authoritative organizations. However, no organization had stands out as the DevOps authority yet.

4.5.1 DevOps Certification

Within IT, there are many different disciplines. Many disciplines have established authorities that set the standards. For example, PMI [113] authorizes project management professional (PMP) certificates bases on the Project Management Book of Knowledge (PMBOK). IIBA [114] authorizes business analysis certificates based on the Business Analysis Book of Knowledge (BABOK). ICAgile [115] is the certification and accreditation body of Agile professionals. However, DevOps does not have similar authoritative organization.

ICAgile is a front-runner in establishing themselves as the DevOps authority. ICAgile is trying to establish a separated track of DevOps certification, in addition to its Agile tracks. ICAgile built its DevOps learning roadmap through community contribution from pioneers, experts, and trusted advisors. ICAgile depends on experienced DevOps practitioners for guidance. Hence, the primary source of DevOps knowledge is from experience.

The ICAgile's DevOps certification path starts with from ICAgile Certified Professional in Foundations of DevOps (ICP-FDO), then ICAgile Certified Professional in Implementing DevOps (ICP-IDO), and finally ICAgile Certified Expert in DevOps (ICE-DO). The expert level is reserved for those who "gain[ed] exposure in the community by writing great blogs, speaking at conferences, and publishing [DevOps] books" [19]. ICAgile describes ICP-FDO as:

> *The ICP-FDO is one of two knowledge-based certifications on the DevOps Track. This certification provides an overview of core concepts for DevOps and is geared towards a broad audience of professionals, technical and non-technical. The learning objectives cover areas such as the* ***business case for DevOps, Continuous Integration, Continuous Delivery, accompanying cultural changes, operational considerations, configuration management****, etc. Participants who complete this certification will gain an excellent foundation in DevOps concepts and ingredients for a successful transition.*
>
> *Typically, training providers will cover the required learning objectives in approximately 14 hours of instructional activities over the course of two days.*
>
> *- ICAgile ICP-FDO [17]*

When I first looked up the objectives of ICP-FDO in December 2017, there were 22 major topics and 95 sub-topics, as listed in **Appendix K ICAgile ICP-FDO Learning Objectives (Dec 2017)**. The long list of topics and sub-topics shows the complexity of DevOps. No training can cover all 22 major topics and 95 sub-topics thoroughly in the recommended two-day period. Therefore, ICP-FDO provides only an overview of DevOps. Students who received ICP-FDO certificates were not recognized as DevOps practitioners. In April 2018, ICAgile published an official Learning Roadmap DevOps Track [116]. In the new roadmap, ICP-FDO learning objectives have been reduced to five major topics with 14 sub-topics. Hence, even the DevOps experts were in the process of learning and defining DevOps.

In December 2017, objectives for ICP-IDO were not yet defined. Hence, it was hard, even for DevOps experts, to define the learning outcomes needed for DevOps practitioners. In the new roadmap published in April 2018, the purpose for ICP-IDO was defined as:

> *The ICP-IDO is one of two knowledge-based certifications on the DevOps Track. This certification takes a **hands-on** approach to planning, building, monitoring, and maturing a DevOps pipeline. While the learning outcomes themselves are **technology agnostic**, course designers will accredit their courses based on technology stacks and tool suites most relevant to their intended course audience. Participants who complete this certification will gain an excellent foundation for implementing DevOps in their organizations and will become well-versed in avoiding common pitfalls and overcoming obstacles to DevOps implementations.*
>
> *- ICAgile ICP-IDO [18]*

There are 4 major topics and 14 sub-topics in the ICP-IDO roadmap. However, no training organization has figured out how to provide training that satisfies the ICP-IDO learning objectives yet. Therefore, the only way to become a DevOps practitioner is still through hands-on working experience.

4.5.2 DevOps Training Programs

For non-technical personnel there are fewer available training programs. In November 2017, I searched for DevOps training on Coursera [21], which provides "universal access to the world's best education". When searching Coursera for courses using the keywords "Continuous Integration" and "DevOps", there were 0 results. Hence, DevOps was not even a valid topic for Coursera.

In July 2018, I searched Coursera using the keywords "Continuous Integration", "Continuous Delivery", and "DevOps" again. The searches returned 22 courses for "Continuous Integration", 16 for "Continuous Delivery", and 13 courses for "DevOps". These searches showed that DevOps gained recognition in eight months between 2017 and 2018. However, none

Topic	DevOps	CI	CD
Agile	2	2	2
Microservices	2	2	2
Cloud	4	2	3
Tool	1	4	1
Not directly related	4	12	8
Total	13	22	16

Table 4.5 Actual Topics on Coursera CI, CD, and DevOps courses

of those courses were actually about CI, CD, or DevOps. Instead, those courses were about Agile, microservices, the cloud, specific tools, and other topics not directly related to DevOps. For example, the course "Teaching Impacts of Technology in K-12 Education" [117] was one of the results for the keyword "DevOps". It does not relate to DevOps directly. Table 4.5 summarizes the search results. Hence, even though DevOps has gained recognition, it is not educated recognition.

In addition, I searched Skillsoft [22] for DevOps courses, since Skillsoft is the global online learning resource for many corporations. Skillsoft returned 32 courses. The search results were similar to Coursera's, with 30 of the 32 courses not directly related to DevOps. Two courses "DevOps Fundamentals: Tools, Technologies, and Infrastructures" and "Integrate Development and Operations (DevOps)" have course hours of one hour and 56 minutes, and one hour and 32 minutes, which provided only a high-level description of DevOps. ICAgile recommends that the ICP-FDO course be two days; Skillsoft's learners could not learn much about DevOps in less than two hours.

As described in Section 4.5.1, ICAgile defined learning objectives for the ICP-FDO certificate. There are multiple IT training providers around the world that offer ICP-FDO training certified by ICAgile, but there is no provider for ICP-IDO training yet. ICAgile provides information about upcoming ICP-FDO classes around the world in different regions. In November 2017, there were 19 upcoming ICP-IDO classes around the world within one year. In July 2018, there were 59 ICP-FDO classes around the world within one year. Among the 59 classes, 46 would be offered in United States of America, including 8 offered online; 8 offered in Australia and New Zealand; 3 in Canada; 2 in Southeastern Asia, but none in the rest of the world. The increasing occurrences of ICP-FDO classes shows increasing demand and interest in DevOps. However, the numbers also show that the interest is regional, concentrated in North America. DevOps education needs to spread more globally.

Among the ICAgile certified ICP-FDO training providers, ASPE Training [20] was the primary provider in the United States of America and online. The ASPE Training course "DevOps Implementation Boot Camp (ICP-FDO)" [118] actually takes three days, instead of the two days recommended by ICAgile. It takes more time to convey even basic DevOps knowledge than the experts estimate. In conclusion, industry lacks DevOps training for both technical and non-technical personnel.

4.5.3 DevOps Communities, Conferences, and Organizations

DevOps is rapidly gaining recognition through the effort of many IT practitioners. DZone Integration Zone [30], DZone DevOps Zone [31], and DevOps.com [32] are significant DevOps communities where people can learn and share information and experiences about DevOps through writing, publications, webinars, and more. DevOps service providers also market their services. In addition to these communities, there are many personal blog sites about DevOps [29]. ICAgile defines Certified Expert in DevOps (ICE-DO) as great DevOps blog writers, conference speakers, or book authors [19]. These communities provide important platforms for DevOps practitioners to publish, and businesses to reach their potential clients, therefore advancing DevOps maturity.

Besides the DevOps communities, there are increasing numbers of DevOps conferences and events around the world. All Day DevOps [23] is an annual online conference that runs continuously for 24 hours, with more than one hundred speakers, and audiences from over one hundred countries in 38 time zones. All Day DevOps offers sessions in five tracks including CI/CD, Cloud-Native Infrastructure, DevSecOps, Cultural Transformations, and Site Reliability Engineering. DevOps World - Jenkins World [24] is organized by Jenkins [78], a DevOps tool producer. It provides "opportunities to learn, explore, network, and help shape the future of DevOps and Jenkins" [24]. DevOps World - Jenkins World has 2500 attendees from all over the globe, and more than one hundred workshops covering software automation, DevOps culture, performance measurement, security, and more. In addition, DevOps Enterprise Summit [25], started in 2014, has more than 1400 attendees and over one hundred speakers. DevOps Enterprise Summit emphasizes insights and exchanges between "practitioners and subject matter experts in DevOps, who are pioneering the philosophies and practices that work (and conversely, which to avoid)" [25]. Finally, there is DevOps Experience Virtual Summit [26], which focuses on experience sharing among DevOps experts. There are many other DevOps events. The many events show that the community is trying to improve DevOps awareness training, and education by gathering experience from IT practitioners, which affirms my potential theory.

In addition to industrial conferences, DevOps is starting to get interest from academic conferences. Centers for Advanced Studies Conference (CASCON) [27], an annual international conference on computer science and software engineering, introduced a CI workshop in 2016 and a DevOps workshop in 2017. International Workshop on Continuous Software Evolution and Delivery (CSED) [28], focuses on CI and CD. It co-located with the 2016 International Conference on Software Engineering (ICSE) [119]. There are also organizations like DevOps Research and Assessment (DORA) [33] that preforms DevOps research, and is trying to institute a "scientific approach to software development, product management, and organizational transformation" [33].

In conclusion, DevOps is still progressing toward maturity, and is on its way to become an IT standard. In the meantime, it heavily relies on IT practitioners' experience. Nonetheless, the foundation work to formalize DevOps education has started. The ICP-FDO program allows practitioners to learn and certify their basic DevOps knowledge. There are strong communities, conferences, and events where people may learn or publish about DevOps. Academics are also starting to show interest in DevOps through scientific research.

4.6 DevOps Practitioners

Since the progress of DevOps heavily relies on the expertise of the DevOps practitioners, I interviewed DevOps experts to learn about their experience. According to DevOps experts, DevOps practitioners need to be Jacks-of-All-Trades with both social and technical skills, and strong architectural knowledge about IT.

4.6.1 DevOps Practitioners Interview Process

During the interviews, DevOps experts were presented with the proposed grounded theory in **Appendix E Potential Grounded Theory**, and then asked the following questions.

1. What do you think about the grounded theory above?
2. How do you become a CI expert?
3. What skills should a CI expert have?
4. Between 0 and 9, with 0 being least important and 9 being most important, how are these factors contributing to your CI expertise?

   A. Education? [0-9]
   B. Years of hands-on experience? [0-9]
   C. Diversity of hands-on experience? [0-9]
   D. Diversity of skills? [0-9]
   E. Knowledge about different tools? [0-9]

5. Others factors important to your CI expertise?

As defined by ICAgile, DevOps experts were people who "gain[ed] exposure in the community by writing great blogs, speaking at conferences, and publishing [DevOps] books" [19]. Therefore, I invited community article writers, conference and webinar speakers, and book authors for interviews.

From DZone DevOps Zone [31], I searched for well "liked" DevOps articles and invited writers for interviews. Since DZone does not provide contact information for the writers, I searched for the corresponding writers in social media, and invited them for interview. Eight writers were invited. Then I searched for speakers in All Day DevOps 2017 [23]. All Day DevOps 2017 had five tracks, I invited keynote speakers and speakers from each track for interviews, 12 in total. DevOps.com [32] offered many DevOps webinars. I invited six webinar speakers for interviews. In addition, I invited two researchers from DORA [33], one instructor from ASPE Training [20], and two DevOps book authors for interviews. Last, but not least, I visited the local Python User Group [120] to invite the local IT practitioners to participate.

In February and March, 2018, I interviewed four volunteer DevOps experts and transcribed the interviews. I applied selected coding on the transcriptions, as in Section 4.3, to verify, reject, or enhance my potential theory. The interviewees provided consistent answers to my questions. Therefore, I determined that there was sufficient data to saturate my theory and seek no addition interviewees.

4.6.2 DevOps Practitioners Interview Analysis

It was not easy to get DevOps experts to participate in academic research through cold calls. I wanted to express special gratitude to two of the DevOps experts, Viktor Farcic, author of the book "The DevOps 2.0 Toolkit" [53] and Alan S. Koch, senior lead instructor of DevOps certification courses at ASPE Training [20]. I had their permission to publish their names. They have provided significant input to my GT project. The other two volunteers were senior IT practitioners with wide and deep IT experience, who were passionate about the future of DevOps.

Factor	Expert1	Expert2	Expert3	Expect4
A. Education? [0-9]	3	0	5	2
B. Years of hands-on experience? [0-9]	7	9	8	9
C. Diversity of hands-on experience? [0-9]	9	6	6	9
D. Diversity of skills? [0-9]	8	5	7	9
E. Knowledge about different tools? [0-9]	7	3	7	0*

* Expert did not provide any rank, but said tools were irreverent.

Table 4.6 Experts Ranked Factors of Becoming DevOps Practitioners.

4.6.2.1 ***DevOps Practitioner Requirement Analysis***

During the interviews, the interviewees ranked the importance of different factors in becoming DevOps experts. Table 4.6 shows the results. All interviewees consistently ranked education the least important factor of becoming a DevOps practitioner. The interviewees indicated that education could only provide fundamental DevOps knowledge. One of the interviewees said that education was useful "only when you're looking for the first job".

Two interviewees ranked knowledge of different tools low. As one of the interviewees said, "What really matter [was] that [practitioners] have enough experience … understand the logic behind … instead of specific skill." Experienced IT practitioners could learn to use any tools as needed. Another interviewee (Expert4 in Table 4.6) did not provide any rank regarding knowledge of different tools. The interviewee considered specific tools were irrelevant to becoming a DevOps practitioner. Therefore, I assigned zero to the factor for the expert. One interviewee ranked knowledge of different tools 7 out of 9, but he remarked that "knowledge in specific tools is not as important as having a broad knowledge of the available tools, [and] how they fit together". Hence, the interviewees disagreed with my potential theory about DevOps practitioners favoring their familiar tools. As one of the experts said, "Someone having very board experience … [could not] be a huge expert in certain [tool]." Experts claimed that DevOps practitioners should not limit themselves to specific tools, but focus on understanding system level architectural concepts.

All experts ranked either years of hands-on experience, or diversity of hands-on experience the most important factor of becoming DevOps practitioners. One interviewee pondered that DevOps practitioners need more than experience. They need diversified experience, where "diversity needs to represent both breadth of experience and depth of experience". It confirmed my potential theory that the DevOps craftsmanship could only be learned through experience, and in this case, diversified experience.

4.6.2.2 *DevOps Practitioner Expectation Analysis*

In addition to the ranking questions, I asked the interviewees the following questions.

What do you think about the potential grounded theory?

They all agreed that DevOps was experience based. As one of the interviewees quoted from Daman Edwards that "there [was] no one true authority on what DevOps [was] or [wasn't], because DevOps [was] an experience-based movement" [121].

The interviewees also agreed that DevOps was "akin to an art form" for there was no "formula or recipe to accomplish [DevOps] goal". Therefore, DevOps was not craftsmanship that IT practitioners learned, and then applied. Instead, IT practitioners applied their experience to achieve DevOps craftsmanship. One of the interviewees further argued that DevOps was not craftsmanship by a single IT practitioner, but by a team. It was about "creating self-sufficient teams" that were capable of carrying out the craftsmanship. Most IT development involved a team. The forefront of DevOps was to handle the social aspect of the development team.

How do you become a CI expert?

One of the interviewees concluded that DevOps practitioners should have "the breadth and depth of IT experience that [would] provide context for all of the DevOps practices and tools".

What skills should a CI expert have?

None of the interviewees named a specific skill, such as Java programming. As one of the interviewees said, "There [was] not a truly DevOps tool or ... skill." Another interviewee summarized that "DevOps expert must have a wide variety of technical, leadership, management, and customer support skills that [came] from actual hands-on experience in a wide variety of IT roles. But among those skills, none [was] so fundamental that it [was] required". What DevOps practitioners needed were high-level skills, such as problem solving and automation. DevOps did not entail any fixed tool, skill, or solution. Therefore, each DevOps implementation was unique as a form of art, as depicted in my potential theory.

What others factors are important to your CI expertise?

One of the interviewees further illustrated that DevOps practitioners need more than skills. DevOps practitioners also needed trust and authority from the higher management. "DevOps [was] about transforming an organization." DevOps implementation often provokes cultural changes and causes fear. IT practitioners need empowerment and trust to overcome the barrier. Therefore, many IT practitioners argue that DevOps is a cultural, procedural, and technological movement [3] [4] [5] [6] [7] [8] [9] [10], with cultural changes overshadowing procedural and technological changes. In conclusion, DevOps experts described DevOps practitioners as individuals with wide and deep hands-on working experience, problem solving skills, and interpersonal skills. In addition to having the right DevOps practitioners, companies wanting to adopt DevOps, should start by embracing the DevOps culture.

In conclusion, my GT project studied DevOps education in depth, from academic and industrial perspectives, through questioning, selective coding, surveying, data gathering, and making comparisons. From my study, a GT theory emerged that DevOps education heavily relies on hands-on working experience, while classroom education provides fundamental DevOps knowledge. With a strong DevOps community already existing, DevOps is on its way becoming an IT standard. However, until a DevOps standard is broadly accepted by the IT industry, comprehensive DevOps education is not likely achievable. In the meantime, institutions should include fundamental DevOps knowledge in education, and increase the opportunity for students to have hands-on practice. I will provide my hypotheses about DevOps education in the next chapter.

5 Hypotheses about DevOps Education

In the previous chapter, the in-depth GT study about DevOps education concludes that DevOps lacks commonly recognized scope, curriculum, and credentials. The roles and responsibilities of DevOps practitioners are still rapidly changing. Therefore, institutions cannot yet provide comprehensive DevOps experience to their students. Instead, academic institutions should provide fundamental DevOps education to prepare students for their future DevOps advancement, and foster DevOps research to benefit academia and industry.

Upon reflection on the outcome, I propose 11 hypotheses with experiments that could ascertain the benefits academia, students, and industry stand to receive by teaching and adopting DevOps. These 11 hypotheses are inspired by the findings of my GT project to confront the shortcomings of DevOps education. Conducting these experiments will help fostering DevOps academic research and teaching. These hypotheses will also help to cultivate the connection between academia and industry.

For each hypothesis, I provide my inspiration, motivation, and goal to achieve. These hypotheses are falsifiable. Therefore, I define concrete falsification conditions and test procedures for them. These 11 hypotheses are divided into five groups, and they are listed in the rest of this chapter.

5.1 DevOps in Programming Assignments

The first group of hypotheses studies whether embedding different DevOps processes into programming assignments will improve student learning outcomes. Educators keep looking for better way to teach computer science. If embedding DevOps processes into teaching will improve student learning outcomes, while at the same time preparing them for the job market, then educators will be motivated to do so.

Hypothesis #1: *Students who employ the continuous code integration pipeline with auto-build and auto-deploy while doing programming assignments get statistically significant higher grades for their programming assignments.*

Inspiration: According to my curriculum search result in Section 4.1, most institutions are not teaching DevOps related topics. Therefore, it is doubtful any DevOps curriculum exists in academia. In addition, most academics are not interested in learning or adopting DevOps (Section 4.2); so it is unlikely that academics will create curriculum for DevOps practices. If there exists DevOps curriculum, which has been proven beneficial to students, then academics may adopt such curriculum to teach DevOps.

Purpose: Continuous integration streamlines many processes, including continuous code integration, continuous database integration, continuous testing, continuous inspection, continuous delivery, and continuous feedback. However, it is not known whether simply streamlining programming processes will improve program quality. If so, better program quality should be reflected in higher grades on programming assignments. Therefore, this hypothesis studies whether streamlining the programming processes in programming assignments improves student learning outcomes. If so, institutions should embed these DevOps processes in students' assignments.

This hypothesis streamlines only code integration, auto-build, and auto-deploy in programming assignments. Programmers usually execute these processes manually. Streamlining these processes adds no direct value to the programs' quality. Therefore, this hypothesis studies whether streamlining the programming processes improves program quality by increasing student understanding and practice of programming.

Falsification: In a computer science course with programming assignments, students in one class do programming assignments with no special instruction. In another class, students employ the continuous code integration pipeline with auto-build and auto-deployment for the same programming assignments. The hypothesis is falsified if students who employ the continuous code integration pipeline in their assignments do not have statistically significant higher grades on average for their programming assignments.

If continuous code integration, auto-build, and auto-deploy do not improve student learning outcomes, then it may not be important for institutions to embed them in assignments.

Hypothesis #2: *Students who implement auto-test while doing programming assignments get statistically significant higher grades for their programming assignments.*

Inspiration: According to my curriculum search results in Section 4.1, most institutions teach software testing. However, it is not clear whether the curriculum is actually teaching automated continuous testing as required by DevOps (Section 2.1), or just software testing concepts. I would like to study automated continuous testing in education explicitly.

Purpose: Among different continuous integration topics, academic education covers testing the most. This hypothesis studies whether automated continuous testing improves student learning outcomes in programming assignments. If so, institutions should include automated testing in programming assignments.

Falsification: In a computer science course with programming assignments, students in one class do programming assignments with no special instruction. In another class, students must implement specified automated tests in addition to the programming assignments. The hypothesis is falsified if students who implemented automated tests do not have statistically significant higher grades on average for their programming assignments.

If automated testing does not improve student learning outcomes, then it is not important for institutions to include it in assignments.

> ***Hypothesis #3:*** *When doing assignments, students who receive continuous feedback regarding their progress throughout implementing their assignments get statistically significant higher grades than other students who submit their assignments only upon completion.*

Inspiration: Duvall *et al.* detailed six CI tasks [39], one of the CI tasks is continuous feedback. During my curriculum search in Section 4.1, I found teaching about the other five CI tasks, but none about continuous feedback. Therefore, I particularly like to study continuous feedback in computer science education.

Purpose: Feedback is an important part of education. One of the continuous integration tasks is continuous feedback. This hypothesis studies whether employing continuous feedback improves student learning outcomes in programming assignments. If so, institutions should employ continuous feedback in programming assignments.

Falsification: In a computer science course with programming assignments, students in one class continuously check their code into a repository, then receive auto-feedback about their progress right away throughout implementing their assignments. Students in another class do the same programming assignments with no special instruction, and submit their assignments only upon completion, then receive human marking and feedback. Compare the grades of the students at the end of the course. The hypothesis is falsified if students who receive auto-feedback do not have statistically significant higher grades on average.

If automated feedback does not improve student learning outcomes, then it is not important for institutions to employ it in assignments.

5.2 **DevOps Skills Transferability**

The second group of hypothesis studies whether DevOps skills are transferable. Hence, whether acquiring DevOps skills improves student learning outcomes in other computer science subjects. If so, it should motivate all computer science educators to adopt DevOps.

Hypothesis #4: *Students, who practice a combination of continuous code integration, auto-build, auto-deploy, and auto-test in their first two software engineering courses, get statistically significant higher grades in senior-level non-software engineering courses with programming components, including computer networks, database management systems, and computer graphics.*

Inspiration: When inviting non-software engineering research groups to participate in DevOps research in Section 4.3, all non-software engineering research groups showed no interest in learning more about DevOps. Therefore, I am interested in studying whether learning DevOps has a positive effect on learning other computer science subjects.

Purpose: Data shows that practicing DevOps improves software quality, but it is not known whether DevOps skills have other benefits. Therefore, this hypothesis studies whether the hands-on experience with the software development pipeline combining continuous code integration, auto-build, auto-deploy, and auto-test enhances students' ability to learn computer science knowledge in non-software engineering areas. If so, institutions should introduce DevOps into the curriculum. Other non-software engineering computer science specializations should also care about DevOps education.

Falsification: In the first two junior-level software engineering courses, half of the students employ the software development pipeline combining continuous code integration, auto-build, auto-deploy, and auto-test to do their programming assignments, while there is no restriction for the other students. Compare the grades of the students in senior-level courses for computer networks, database management systems, and computer graphics. The hypothesis is falsified if students who employed the software development pipeline in junior-level software engineering courses do not have higher grades than those who did not employ the pipeline, and the difference is statistically significant.

5.3 **DevOps Academic Values**

The third group of hypotheses studies whether DevOps adds value to academic publications and can initiate joint DevOps research with computer science academics in non-software engineering specializations. I will also use this study to foster academic awareness of DevOps.

> ***Hypothesis #5:*** *Compared to research papers without artifacts, research papers with "Reproducible" artifacts published in conferences, like Foundations of Software Engineering (FSE), receive statistically significant higher number of citations within the first five years of publication, since the ability to create "Reproducible" artifacts is a major benefit of DevOps.*

Inspiration: During my graduate student interviews in Section 4.3, graduate students expressed their concern that adopting DevOps took a lot of effort, but academia did not reward the effort appropriately. Therefore, it is not worth the effort. I am interested in studying whether DevOps can increase the value of academic publication.

Purpose: For research projects with programming components, DevOps allows other researchers to regenerate experiment results anytime and anywhere. Others researchers can easily build on top of the existing results. This hypothesis studies whether DevOps' ability to regenerate experimental results increases publication value in term of citation-count. If so, it should be a strong reason for academics to learn and adopt DevOps.

Falsification: Research papers with artifacts, accepted by conferences like FSE, are given artifact badges, such as Functional, Reusable, Available, Replicated, Reproduced [122]. Five years after the publication, count the number of citations for each paper with badges like Reproduced in proceedings. Compare the average number of citations for papers with

Reproduced badges and the proceeding's citation impact factor [123]. The hypothesis is falsified if the average number of citations for papers with the Reproduced badge is not statistically significant higher than the proceedings' citation impact factor.

Hypothesis #6: *Students who employ continuous inspection with artificial intelligence while doing programming assignments get statistically significant higher grades for their programming assignments.*

Inspiration: During my graduate student interviews in Section 4.3, graduate students identified barriers adopting DevOps, including lack of knowledge, a steep learning curve, too much trial-and-error, unfriendly tools, and lack of technical support. If a DevOps platform is readily available for academics to evaluate their hypotheses, would academics be more interested in DevOps?

Purpose: Continuous monitoring and feedback is important in DevOps. Incorporating artificial intelligence (AI) into continuous monitoring and feedback may "detect anomalies, predict performance problems and deviations from the baseline, suggest optimizations, correlate signals across multiple platforms for troubleshooting, do root cause analysis and even automate fixes" [124]. Therefore, there are many opportunities to initiate joint research between DevOps and AI. Joint research is one way to create DevOps interest among academics in non-software engineering specializations.

Falsification: In a computer science course with programming components, all students employ the pipeline of continuous code integration, auto-build, and auto-deploy for their programming assignments. Half of the students also embed continuous inspection with AI in the pipeline, and the other half do not. The hypothesis is falsified if students who embed continuous inspection with AI in the pipeline do not score statistically significant higher on average in their assignments.

5.4 DevOps Employment Opportunities

The fourth group of hypotheses studies whether knowing DevOps improves students' employment opportunities and speeds up job integration. If so, it should motivate students to learn DevOps.

Hypothesis #7: *Students who practice a combination of continuous code integration, auto-build, auto-deploy, and auto-test while doing programming assignments have a higher chance to be employed.*

Inspiration: The job postings search in Section 4.4 shows that there is a high demand for DevOps practitioners in industry. Data in Section 4.5 also shows that DevOps is becoming a standard requirement for IT practitioners. Therefore, I am interested in studying whether DevOps education gives students an advantage in industry.

Purpose: There is a large demand for DevOps practitioners in industry. This hypothesis studies whether practicing a combination of continuous code integration, auto-build, auto-deploy, and auto-test allows students to answer interview questions better, therefore increasing their employment opportunities.

Falsification: In one class, students do programming assignments with no special instruction. In another class, students employ the DevOps pipeline combining continuous code integration, auto-build, auto-deploy, and auto-test to do the same programming assignments. Cooperate with industrial internship employers to score (0-100) students whom they interviewed for the internship positions. Ask the industrial internship employers whether their company uses DevOps. Use factor analysis to compare the scores from employers using and not using DevOps versus students who employed the DevOps pipeline and those who did not. The hypothesis is falsified if students who employed the DevOps pipeline in their assignments do not score statistically significant higher on average in the interviews for both employers using and not using DevOps.

Hypothesis #8: *Students who practice a combination of continuous code integration, auto-build, auto-deploy, and auto-test while doing programming assignments value the course more when adapting to industrial work.*

Inspiration: The last hypothesis studies whether DevOps education gives students an advantage from the employers' perspective. I am also interested in studying whether DevOps education gives students an advantage from the employees' perspective.

Purpose: This hypothesis studies whether employing the pipeline combining continuous code integration, auto-build, auto-deploy, and auto-test in programming assignments enhances computer science education, so that students will value the course more when joining the industrial work force.

Falsification: In one class, students do programming assignments with no special instruction. In another class, students employ the DevOps pipeline combining continuous code integration, auto-build, auto-deploy, and auto-test to do the same programming assignments. After the students come back from their industrial internship programs, ask the students to score (0-100) the usefulness of the course during their internship. The hypothesis is falsified if students who employ the pipeline in their assignments do not score the course statistically significant higher than those who did not employ the pipeline.

5.5 DevOps in Industry

The fifth group of hypotheses is more industry oriented. Since the DevOps movement started from industry, DevOps education should stay connected with industry. These hypotheses relate industrial DevOps experience with academic education.

Hypothesis #9: *IT practitioners with industrial working experience, can learn DevOps skills faster than students without industrial working experience.*

Inspiration: In Section 4.6, while selective coding the DevOps expert interviews, the code "hands-on experience" kept coming up. However, hands-on experience can mean many different things. It can mean hands-on experience in using specific tools, or hands-on experience in directly supporting customers. Therefore, I would like to learn more about the substance of the term "hands-on experience".

Purpose: DevOps experts believe that IT practitioners need hands-on experience to be proficient in DevOps. When DevOps experts refer to hands-on experience, they may refer to work experience, instead of DevOps usage experience. This hypothesis studies whether working experience actually benefits DevOps learning. If so, educators should further study how to incorporate work experience into academic education.

Falsification: Provide DevOps lectures to IT practitioners (with a minimum 3 years of industrial working experience) and students without work experience, where neither the IT practitioners nor the students have previous DevOps knowledge. Then specify the IT processes and environment of a company, and ask the IT practitioners and students to describe what they will do to setup a DevOps pipeline in the company. The descriptions are scored according to their correctness and completeness. The hypothesis is falsified if the IT practitioners do not have statistically significant higher scores on average then the students.

> ***Hypothesis #10:*** *Companies adopting DevOps culture (acceptance of development-operations as one team, willingness to fail fast, experiment with new technology, and focus on quality [5]) have a higher employee satisfaction rate then companies only adopting DevOps technology (procedures and tools).*

Inspiration: DevOps literature review in Section 2.2 shows that many IT practitioners consider DevOps a culture movement. However, open coding in the first phase of this GT project (Section 3.3) shows that IT practitioners have relatively low interest in DevOps culture. I am interested in studying whether DevOps is really a cultural movement.

Purpose: Many DevOps practitioners stated that DevOps is more a cultural adoption than technical adoption. This hypothesis studies whether companies aiming at adopting DevOps culture, instead of DevOps technology have better employees' satisfaction rate. Companies that adopt DevOps as a culture foster acceptance of development-operations as one team, willingness to fail fast, experiment with new technology, and focus on quality [5]. Then all changes are implemented to suit the DevOps culture. Companies that adopt DevOps technology look for DevOps procedures and tools that benefit the companies, and implement changes in the existing environments. If DevOps culture is more important than DevOps technology for success, then educators should study how to incorporate DevOps culture into academic education.

Falsification: Among companies going to adopt DevOps, provide half of the companies training about how to adopt DevOps through cultural changes, and the other half about how to adopt DevOps through technological changes. One year later, survey the companies' workers, involved in the DevOps adoption, about their satisfaction rate (0-100) of adopting DevOps. The hypothesis is falsified if companies adopting the DevOps culture do not receive statistically significant higher satisfaction rate on average than companies adopting the DevOps technology.

Hypothesis #11: *Companies using Agile benefit more from adopting DevOps than companies using non-Agile project management methodologies, in term of key performance factors, such as deployment frequency, deployment lead-time, and mean time to recover.*

Inspiration: During the curriculum search in Section 4.1, most of the DevOps topics are covered in senior software engineering courses. Many of these senior software engineering courses also cover Agile project management. In addition, ICAgile is trying to establish a track of DevOps certification (Section 4.5). Therefore, I am interested in studying whether DevOps complements Agile.

Purpose: DevOps is a cultural, procedural, and technological movement that affects the software development process. Agile (including SCRUM, XP, etc.) is a project management methodology currently drawing the most attention from the IT industry. Therefore, this hypothesis studies whether DevOps benefits companies using Agile more than DevOps benefits companies using non-Agile project management methodologies. If so, educators should further study how to incorporate DevOps into their current project management methodology teaching.

Puppet used deployment frequency, deployment lead-time, and mean time to recover as benchmarks to identify high-performing DevOps teams [14]. Therefore, this hypothesis uses the same criteria to measure improvement from DevOps adoption.

Falsification: Companies that are going to adopt DevOps should calculate their deployment frequency, deployment lead-time, and mean time to recover as a benchmark before the adoption. Then they should calculate their deployment frequency, deployment lead-time, and mean time to recover one year after adopting DevOps. The hypothesis is falsified if after one year, companies using Agile do not have statistically significant higher improvement on average in all three key performance factors than companies using non-Agile methodologies.

5.6 Concluding Remarks

These hypotheses can be used to study whether embedding DevOps processes in programming assignments will enhance the student learning outcomes; examining whether DevOps skills are transferable; exploring how to encourage academic DevOps research; evaluating how DevOps knowledge affects student employment opportunities; and last but not least, investigating how to connect academic DevOps learning with industrial working environments. Studies based on my 11 DevOps education hypotheses could help illuminate the path and direction for DevOps education in academia and industry.

6 Limitations

The 11 hypotheses in the previous chapter conclude the contributions of my GT project. When using GT, a social sciences qualitative research methodology, I encountered different challenges from using traditional hypothetico-deductive methodology. I describe my limitations during the study in this chapter.

In 1967, Glaser and Strauss first described GT [90], a social sciences qualitative research methodology. GT is an inductive methodology, which is very different from the traditional hypothetical-deductive methodology. GT uses "nonmathematical process of interpretation … to discover concepts and relationships in raw data and then organize these into theoretical explanatory scheme" [89]. Since 1967, GT has been "prove[n] an extremely useful research approach in several fields including medical sociology, nursing, education, and management theory" [91]. Many software engineering publications report the results of GT projects [91]. Hence, GT is a well-accepted methodology for software engineering research. Even though GT is a well-accepted methodology, it has some limitations, which I will discuss in this chapter.

6.1 Internal Threats

Since GT was originally a social sciences qualitative research methodology, some computer science researchers may have never heard about it, or have difficulty understanding the GT processes and values. To address this, I added descriptions of the GT processes throughout the dissertation to ensure that the readers better understand the progress and findings.

GT uses open coding to identify key topics of interest from field stakeholder interviews. Some may argue that open coding is a personal interpretation of the interview transcriptions. Therefore, the validity of open coding is questionable. To address this concern, each interview transcription was open coded by two investigators. Through comparison and discussion, I tried my best to eliminate misinterpretation and personal opinion from the open coding result. I could have involved more investigators in open coding to increase the credibility of the results. When open coding the four interview transcriptions individually, both investigators identified the same phases for the same concepts almost all the time. When comparing the individually identified codes, the investigators mainly discussed about the rhetoric of the codes. Considering the relatively consistent coding results between the two investigators, adding another investigator would not add significant value. Therefore, I did not involve additional investigators during open coding.

6.2 External Threats

If all the interviewees and data are from the same underlying population (e.g. restricted geophysical region), a GT project may have external validity problem. Hence, the same GT project may generalize different findings with different underlying populations. I tried my best to break the localization barrier. I searched the computer science curricula of institutions around the globe. I invited academics to my survey from institutions globally. I searched for DevOps certification, training, conferences, and communities internationally. In addition, I invited DevOps experts for interview worldwide. All these were done to ensure external validity.

Unfortunately, my survey, which invited academics from institutions globally, did not receive a significant number of responses. I considered sending invitations to individual computer science academics. However, in many institutions computer science academics were listed among academics of other disciplines, who could not be clearly identified. Alternately, I could expand my invitation to other institutions, or broaden invitation through social media. However, I doubted the number of responses I would get after expanding the invitation would significantly increase. Therefore, I decided to start with questioning local academics, to find out how I might motivate them to participate in DevOps research, especially among non-software engineering research groups. I planned to meet local academics to have in-depth discussions about their interest in DevOps. In the end, all research groups rejected my invitation. Individual graduate students who accepted my invitation all strongly rejected DevOps adoption. With the strong response from local academics and graduate students, I generalized that academics lacked interest in DevOps without further investigation.

Besides the localization concerns, computer science researchers may also be concerned about the reliability of GT projects. I generalized from interviewing DevOps experts that IT practitioners could only advance their DevOps knowledge and skills through hands-on working experience. To ensure the result is reliable, I selected interviewees with different backgrounds, knowledge, work experience, and who lived in different part of the world. I could have gone back to the interviewees to reconfirm my findings. Instead, I considered that the interviewees, who were from diverse circumstances, provided very consistent answers; they had already verified each other answers. The finding should be highly reliable, and I did not question the interviewees again.

6.3 **Other Threats**

Studying a fast growing, immature topic like DevOps also poses a threat to data reliability. There are new material, concepts, and events added to the pool every day. I had to stay close to all the latest developments and keep myself up to date. For example, when I started the GT project, I used the term CI. In the middle of the project, I shifted to the term CD. Finally, I selected the term DevOps for my final report. DZone's DevOps reports went through a similar shift. In 2016, DZone named their DevOps report "Continuous Delivery" [57]. In 2017, DZone named their DevOps report "DevOps Continuous Delivery and Automation" [58]. In 2018, DZone named their DevOps report "DevOps Culture and Process" [5].

DevOps rapidly changing trends threaten the reliability of my gathered data. Usually, repeating the data collection procedures to reconfirm the data can increase reliability. However, this may not be the case for DevOps that changes so rapidly. XebiaLabs has to keep refreshing its Periodic Table of DevOps Tools [68] to include the latest inventions. If I repeat my DevOps job posting search, the job titles I used then, may not be valid now. Between December 2017 and April 2018, ICAgile has rewritten their DevOps training objectives for their certificates. If I repeat my DevOps certificate search, a new DevOps authority may emerge. Similarly, there are new DevOps blogs, training, communities, conferences, and organizations that appear every day. Repeating the data gathering process adds minimum value. Instead, I was involved in many of the DevOps communities to stay up to date with the latest DevOps development. I am confident that my reported data aligns with the trends.

Finally, I am interested in studying different education-related topics in computer science. However, there is no corresponding platform for me to gather the required data. Multi-institutional cooperative research on computer science education is very rare. I tried to survey institutions internationally, but could not get the attention from other computer science educators. To improve the validity of future research, it would be nice to establish a platform that gathers computer science education data from institutions.

Even with these limitations, I made contributions to DevOps education, which I summarize in the next chapter.

7 Conclusion

DevOps is a popular movement in the IT industry. XebiaLabs claimed that DevOps is one of the hottest technology search terms in Google Search [1]. In the late 90's, the beginnings of DevOps started as frequently checking source code into repositories. In 2007, Duvall *et al.* officially established CI [39]. In 2010, Humble *et al.* officially defined CD [43]. In 2009, Allspaw and Hammond proposed the DevOps movement that expanded CI and CD from development to operations.

I confirmed the popularity of DevOps by reviewing and following the large number of DevOps blogs, webinars, survey reports, books, conferences, communities, and organizations in industry that tried to understand, promote, assist, and improve DevOps. Because DevOps is popular, many tools are created to support DevOps [68], and new DevOps tools keep sprouting from industry. On the other hand, I searched for DevOps related academic publications in IEEE Xplore, ACM Digital Library, and Google Scholar, and found them relatively scarce. Hence, academia was not as keen on DevOps as industry. To promote academic research in DevOps, I decided to study DevOps.

Since DevOps is a cultural, procedural, and technological movement, I decided to use GT, a social sciences qualitative research methodology, to study DevOps. Using GT allowed me to study both social and technical aspects of DevOps. However, there are many different GT variants. I studied two popular GT variants in depth, and found that the Strauss and Corbin GT [89] was more suitable for my DevOps study.

Following the Strauss and Corbin GT, I started by interviewing five field stakeholders regarding DevOps. Using open coding, I microanalyzed four of the interviews, and finalized 34, 83, 28, and 125 codes from the interview transcriptions. By further analysis, I structured codes into hierarchical trees, and identified seven categories of DevOps interests: "experience", "practice", "benefit", "cost", "tool", "social aspect", and "third-party service", along with 119 potential research questions. In addition, I analyzed the codes to identify shared DevOps interests among the stakeholders. Among the shared DevOps interests, I used data elimination to select DevOps education as my topic of in-depth GT investigation.

Using GT, I studied DevOps education from academic and industrial perspectives. From the academic perspectives, I reviewed the computer science undergraduate curricula of the top 50 institutions, listed in 2017 QS World University Rankings by Subject - Engineering and Technology [101], to determine whether academic education conveyed DevOps knowledge. Among different DevOps topics, 12.8% of the institutions covered CI, 12.8% covered building application, 28.2% covered repository, 23.1% covered deployment, and 89.7% covered testing. Hence, most of the world top institutions did not convey DevOps knowledge to their students, except testing.

I tried to study why academics rarely taught DevOps in undergraduate education, while DevOps was extremely popular in industry. I tried to survey academics globally to validate whether institutions taught DevOps in undergraduate education. However, I did not receive a significant number of responses. Instead, I tried to question local academics, in different research groups, regarding DevOps, but they all declined because they were not interested in DevOps. In the end, I interviewed three graduate students in different research groups to study the reasons for the lack of interests in academia.

The graduate students all agreed that DevOps could improve the quality of their research projects, which involved software development. DevOps could also make their research results repeatable and verifiable by the community. However, these benefits were not enough to motivate DevOps adoption. The graduate students did not believe DevOps would increase their publication opportunities. Therefore, they felt it was not worth their effort to learn and adopt DevOps. If academics are not interested in learning and adopting DevOps, it is not likely that they will teach DevOps.

From the industrial perspective, I searched Monster, DZone job site, and IEEE Computer Society Job Board to study what the DevOps jobs were, and how much demand there was. My results showed that DevOps practitioners were in high demand, but their job titles, requirements, roles, and responsibilities were highly diversified.

Further study of the DevOps job requirements showed that DevOps practitioners were expected to be Jacks-of-All-Trades, who had perfect cultural, procedural, and technologic knowledge and skills. Such DevOps practitioners were rare assets in industry; there were not enough to satisfy the high demand. To fulfill the demand, employers just asked for skills that they thought DevOps needed, without actually understanding DevOps. Therefore, job postings for DevOps practitioners had very different requirements. DevOps education could not train practitioners to be Jacks-of-All-Trades, or train practitioners to fulfill undefined requirements.

The roles and responsibilities of a profession are usually defined by an authoritative organization. Therefore, I tried to search for a DevOps authority. However, DevOps was in its early stage of formalization, so no such authority existed. ICAgile was one of the organizations working on defining learning objectives for DevOps certificates. However, ICAgile's DevOps learning objectives changed dramatically from version to version. DevOps education is a rapidly changing target that is hard to identify.

Since DevOps education is a moving target, I searched for existing DevOps training in industry to determine what IT practitioners could learn from the available training. I searched ICAgile's certified training providers, Coursera, and Skillsoft for DevOps training. My results showed that there was a limited amount of DevOps training for technical practitioners, and what was available was mostly in North America. For non-technical practitioners, DevOps training was almost non-existent. Even with the available training, the validity of the content was questionable.

Since DevOps training was limited, I searched for DevOps learning resources available to the public. From the DevOps communities, I found many DevOps blogs, webinars, industrial research reports, organizations, conferences, and books, but no commonly recognized authority or credential. However, with the large amount of effort from the DevOps communities, an accredited DevOps curriculum may be defined in the near future.

The DevOps communities are relying on the DevOps experts to determine the proper DevOps curriculum. Therefore, I questioned DevOps experts for their insights. According to the DevOps experts, DevOps education could only provide fundamental DevOps knowledge. On top of the basic DevOps knowledge, DevOps practitioners should have solid architectural knowledge and problem solving skills. With this base, DevOps practitioners could advance their DevOps skills through hands-on working experience, diversified experience with both breadth and depth. Adding communication skills, teamwork, and support from management would complete the training of a DevOps practitioner.

After studying DevOps education from the academic and industrial perspectives, a GT theory emerged that DevOps education heavily relies on hands-on working experience, while classroom education provides fundamental DevOps knowledge. With a strong and active

DevOps community already set up, DevOps is on its way becoming an IT standard. However, until a DevOps standard is broadly accepted by the IT industry, comprehensive DevOps education is not achievable.

My findings showed that academic institutions cannot provide comprehensive DevOps training. Instead, academic institutions should provide fundamental DevOps education to prepare students for their future DevOps advancement, and satisfy the high demand of DevOps practitioners in industry. Contributing to that effort, I propose 11 hypotheses about DevOps education in five groups to initiate DevOps education in academic institutions.

The first group studies whether embedding DevOps processes in programming assignments will enhance student learning outcomes. The second group examines whether DevOps skills are transferable. The third group explores how to encourage academic DevOps research. The fourth group evaluates how DevOps knowledge affects student employment opportunities. The fifth group investigates how to connect academic DevOps learning with industrial working environments. In the future, researchers can work on these hypotheses to benefit DevOps education in institutions.

As described in Chapter 2, DevOps is a really broad subject that is popular in industry, but lacks attention in academia. This GT project on DevOps education has only made a small contribution on one of many DevOps topics. There are many other DevOps topics worth studying. I describe some of my potential future DevOps research projects below.

7.1 **Future Work**

This GT project has only studied one of many DevOps topics of interest to IT practitioners. In Section 3.3.2, I described why I selected the "experience" category for further GT study, over other categories, "practice", "benefit", "cost", "tool", "social aspect", and "third-party service". These other categories are also valuable for GT study, but they are not as ready as "experience". In particular, the study of "practice", "benefit", "cost", and "tool" would be more appropriate after the common DevOps practices are defined. Therefore, the next step in my DevOps research is identifying common DevOps practices. DevOps practices have two main aspects: tools and skills. I will create two projects to study DevOps tools and skills.

DevOps employs many tools. XebiaLabs categorized DevOps tools into 15 categories by their functionalities as Source Code Management, Repository Management, Database Management, Configuration Management or Provisioning, Build, Testing, Continuous

Integration (CI), Deployment, Release Management, Logging, Business Intelligence or Monitoring, Cloud or IaaS or PaaS, Containerization, Collaboration, and Security [68].

It is important to prioritize these categories of tools from the industrial perspective, so that researchers can focus on the high priority tools first. To identify the high priority DevOps tools, I will survey high-performance DevOps teams in industry to learn which categories of DevOps tools, identified by XebiaLabs, they employ in their DevOps practice.

I will survey high-performance DevOps teams, because they have successfully implemented and benefited from DevOps. In the 2015 State of DevOps Report [14], Puppet Labs identified high-performing DevOps teams by their high deployment frequency, short deployment lead-time, low failure rate, and rapid recovery time. High-performing DevOps teams deployed code 30 times more frequently and 200 times faster than their peers did. In addition, high-performing teams had 60 times fewer failures and recover 168 times faster. My survey will focus on these teams.

A category of DevOps tools is considered high priority, if over 75% of the high-performance DevOps teams employ that category of tools. With the results, future research can focus on the practices, costs, and benefits of the high priority DevOps tools. In addition, academic institutions can teach students about the DevOps tools according to their priority. Institutions cannot teach students all 15 categories of tools, since computer science curriculum is already pushing the four-year boundary.

From the skills aspect, DevOps does not have a commonly recognized credential or curriculum; therefore, there is no defined DevOps skillset. Skills required for DevOps are very broad. Lee searched through DevOps practitioner job postings and identified eight groups of required skillsets corresponding to the eight groups of tasks below [106].

1. Design, build, and operate technology stack [Operational]
2. Configure, monitor, and manage systems [Operational]
3. Program different tools in various languages [Developmental]
4. Script Linus/Unix processes with various languages [Developmental]
5. Operate different cloud products [Technological]
6. Automate IT processes regardless of tools [Procedural]
7. Understand and enforce best practices [Procedural]
8. Being proficient in interpersonal skills [Cultural]

These eight groups of skillsets thoroughly cover cultural, procedural and technological (developmental and operational) aspects of software development. It is important to prioritize these skillsets from the industrial perspective, so that researchers can focus on the high priority skillsets. To identify the high priority DevOps skillsets, I will survey high-performance DevOps teams to determine which DevOps skillsets, identified by Lee, they use in their daily work.

A DevOps skillset is considered high priority, if over 75% of the high-performance DevOps teams employ the skillset in their daily work. With the results, future research can focus on the effects of these high priority DevOps skillsets. In addition, organizations can use these priorities to define training curriculum and credentials for DevOps practitioners. Academic institutions can also prioritize curriculum changes to match the priority skillsets.

These two projects that prioritize DevOps tools and skills will set the foundation for many other DevOps research in the future. Besides DevOps tools and skills, the "social aspect" of DevOps is another area with many research opportunities, since not much work has been done on it. Researchers may also consider other DevOps research questions listed in **Appendix D Potential DevOps Research Questions**. With the popularity of DevOps still growing fast in industry, any DevOps research will have positive influence in both academia and industry.

8 Continuous Maintenance (CM)

In Section 2.1, I provide an introduction about DevOps. DevOps extends CI and CD from development to operations. Many others proposed combining other aspects of software development into DevOps, including Continuous* [46], DevSecOps [47] [48] [49] [50], ChatOps [51], GitOps [52], DevOps 2.0 [53] and more. I found that neither DevOps, nor these alternatives covered the perspective of continuously maintaining the DevOps environments. I filled the gap by writing the paper "Continuous Maintenance", which identified the continuous maintenance processes not mentioned anywhere.

Some argue that the purpose of DevOps is to eliminate software maintenance from the traditional software development cycle. Therefore, the term continuous maintenance is inappropriate. A better term may be continuous continuity. Since "Continuous Maintenance" is the name of the publication, I kept it as-is in its original form.

The paper "Continuous Maintenance (CM)" [34] was published in the Proceeding of 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME). The paper is reprinted in this chapter, with permission from Candy Pang and Abram Hindle @2016 IEEE.

The major contribution of the paper is proposed the Continuous Maintenance (CM) concept, which is not covered in other literatures, as:

Maintain repositories and artifacts properly and consistently through automation, summarization, compaction, archival, and removal during pre- and post-production.

In addition, continuous maintenance processes are defined for the two phases below.

Pre-Production Phase:

- Repository and storage summarization, compaction, defragmentation, and archival
- Management systems automation

Post-Production Phase:

- Exception or violation analysis
- Temp data and logs summarization and archival
- Data elimination
- Data warehouse and analytics

Since industry is still working on the scope and boundary of DevOps, defining the needs of continuous maintenance will ensure future DevOps advancement, included cultural, procedural, and technical, takes maintenance into consideration.

For this publication, I was responsible for the problem identification, data collection and analysis, as well as the manuscript composition. Dr. Abram Hindle was the supervisory author, and was involved with concept formation and manuscript composition.

Impact: As of writing, Continuous Maintenance has been cited three times according to the Google Scholar [85], including an empirical conference paper that studied continuous software engineering practices [125], and a Master thesis that studied tools and practices to enhance DevOps core values [126].

Abstract

Abstract—There are many "continuous" practices in software engineering, for example continuous integration (CI), continuous delivery (CD), continuous release (CR), and DevOps. However, the maintenance aspect of continuity is rarely mentioned in publication or education. The continuous practices and application execution use many repositories and artifacts, such as databases, servers, virtual machines, storage, data, meta-data, various logs, and reports. Continuous maintenance (CM) seeks to maintain these repositories and artifacts properly and consistently through automation, summarization, compaction, archival, and removal. For example, retaining builds and test results created by CI consumes storage. An automated CM process can remove the irrelevant artifacts and compact the relevant artifacts to reduce storage usage. Proper CM is essential for applications' long term sustainability. There are two sides of CM: pre-production and post-production. During the pre-production phase, CM maintains the health of the development environments and the relevant processes. Then during the post-production phase, CM maintains the health of the applications. This paper identifies some of the CM processes observed in companies.

8.1 Introduction

Continuous maintenance (CM) is an extension of *continuous integration* (CI). The CI concept was first described as "do everything in parallel, with frequent synchronizations" in the 1998 book Microsoft Secrets [41]. Microsoft employs multiple teams to implement different features in parallel, and relies on CI to keep software development synchronized. Companies have been practicing CI for at least two decades. Normally integration happens from end to end, and in practice it is not divided into *continuous integration* (CI) [39], *continuous release* (CR), and *continuous delivery* (CD) [39]. Therefore, the authors combine CR, CD, and other continuous practices into CI for discussion in this paper.

CM refers to the processes that maintain applications' long term sustainability during development and after production. For example, during development, many CI processes rely on version control repositories, and these repositories need maintenance. The CI build process creates artifacts such as executables, meta-data, and various logs. These artifacts need maintenance. The CI test process creates artifacts such as virtual machines, containers, database instances, test data, test results, and various kinds of logs. These artifacts also need maintenance. In production, applications create artifacts such as data and execution logs that need maintenance. CM maintains these artifacts through automation, summarization, compaction, archival, and removal. For example, an automated maintenance process can analyze and summarize execution logs, then compact and archive the logs for a retention period, before removing the logs.

Imagine an application that creates temporary files during a process. The temporary files should be retained for a period of time, then be eliminated under normal circumstances. In this case, the programmers implement the source code that creates the temporary files. The operational analysts allocate the storage where the application writes the files. The programmers specify the life span of the temporary files and any exceptional conditions. An automated CM process will monitor the temporary files, compress the files to use less storage during the retention period, and eliminate the files under normal circumstances. The CM process will also watch for exceptional conditions, and carry out predefined handling procedures. Furthermore, the CM process can analyze and summarize the temporary files to provide additional knowledge. Likewise, CM can maintain bug reports, crash reports, and other non-source code artifacts created by applications.

As IT practitioners, the authors have been observing the end-to-end CI practices in enterprises for more than 10 years in the fields of health care, law enforcement, Internet service, customer management, e-Commerce, telecommunication, finance, investment, and environment. In the IT industry, programmers have embraced many of the CI practices. However, the authors observed that the maintenance processes are often overlooked. Many of the maintenance processes are retrofitted for development or in production after encountering sustainability issues. For example, no CI publication has introduced any continuous process to clean up repositories. When companies deploy CI, they should deploy CM at the same time to maintain CI. By describing various CM processes, the authors hope to raise discussion and awareness about application maintenance. In this paper, the authors describe continuous maintenance (CM) as the continuous processes that maintain development (pre-production) artifacts, and operations (post-production) artifacts through automation, summarization, compaction, archival, and removal.

The remainder of this paper is structured as follows. Section 8.2 reviews existing continuous practices. Section 8.3 lists CM processes. Section 8.4 concludes the paper and sketches future work.

8.2 Related Work

In the existing literature, the maintenance processes for applications' long term sustainability are rarely mentioned. Fitzgerald *et al.* [127] tried to list all continuity related activities from business strategy to development, which "include continuous planning, continuous integration, continuous deployment, continuous delivery, continuous verification, continuous testing, continuous compliance, continuous security, continuous use, continuous trust, continuous run-time monitoring, continuous improvement (both process and product) … [and] continuous innovation". Shamieh [128] added continuous engineering. However, the maintenance aspect of continuity is missing. Some documented continuous practices among practitioners are described below.

8.2.1 Development Management

During development, continuous practices have been well defined. In the book "Continuous Integration", Duvall *et al.* [39] have thoroughly described the following CI practices and their corresponding tools:

- Continuous code integration / continuous build
- Continuous testing
- Continuous inspection
- Continuous feedback
- Continuous delivery
- Continuous database integration

Continuous code integration is the core of the continuous development process. Continuous code integration frequently builds applications from repositories, so that integration problems can be identified early. There are many automated build tools, such as Ant [129]**, Travis CI** [76]**, and Jenkins** [78]**.**

For large projects, compiling a complete build can be very time consuming. A hierarchical build process can be employed to reduce the building time of individual components.

Continuous build can identify integration problems, and **continuous testing** can identify implementation errors. After each build, predefined unit tests and regression tests will be run to identify implementation errors. Regression tests can also be organized into hierarchy to reduce testing time and lower resource consumption. There are many automated testing tools, such as JUnit [130] and other "xUnit" products.

Testing can increase programmers' confidence in their source code. However, having integrated source code that passes testing ensures neither code integrity, nor bug free status. Therefore, **continuous inspection** evaluates the quality of the source code after each change, and looks for potential issues according to previous knowledge.

Continuous inspection uses a variety of tools to examine different source code aspects. For example, Checkstyle [131] makes sure source code adheres to code style standard, EMMA [132] measures code coverage, and FindBugs [133] analyzes code to look for potential bugs. In addition, there are tools that identify code duplication, measure unit tests coverage, estimate energy consumption, and more. Overall continuous inspection safeguards source code quality.

Continuous code integration, continuous testing and continuous inspection can all identify problems, but they rely on **continuous feedback** to alert the responsible personnel.

Continuous delivery orchestrates deployment in different environments, such as individual machines, local servers, or the cloud. Humble *et al.* [43] in their book "Continuous Delivery" describe the continuous delivery practice in details. However, they did not consider the maintenance of artifacts described in this paper.

Last but not least, **continuous database integration** rebuilds databases and test data according to changes in repositories [39]. Continuous database integration is necessary to support continuous code integration and continuous testing.

Many continuous practices have been identified in software engineering. However, they have not considered maintenance processes. DZone has published a Continuous Delivery

Maturity Checklist [134], which does not take into account artifacts maintenance either. Programmers learn how to practice CI through these publications, which do not talk about maintenance, but maintenance is essential to sustain applications. The authors believe that programmers should learn about CM as they learn about CI.

8.2.2 Operations Management

In large companies, applications are implemented by the development team, but run by the operations team [135]. The operations team follows pre-defined IT Service Management (ITSM) activities to maintain quality IT services, separately from the development team or the development process. A few popular ITSM frameworks are listed below:

- *Information Technology Infrastructure Library* (ITIL) [136]
- *Control Objectives for Information and Related Technologies* (COBIT) [137]
- International Service Management Standard for IT Service Management (ISO/IEC 20000) [138]

These frameworks identify processes, roles and responsibilities for IT practitioners to sustain production environments and increase uptime. These frameworks promote circular processes to continuously improve IT services. However, these frameworks do not specify application maintenance processes, or promote coordination between programmers and operational analysts.

It is generally known among IT practitioners that "there was, is, or has been a problem within corporate IT departments between Development and Operations" [139]. DevOps [135] was introduced "as a culture, movement or practice that emphasizes the collaboration and communication of both software programmers and other IT professionals while automating the process of software delivery and infrastructure changes" [6]. DevOps focuses on automating infrastructure, but the literature is not explicit about maintaining applications in operation. CM covers maintenance processes necessary to sustain applications in operation.

CM can also benefit DevOps by purging containers and virtual machines that are not needed anymore. For example, a system may use Docker containers [79] for testing and deployment. In some cases, the containers are made, but left behind. An automated CM process can clean up dangling containers to maintain resource availability.

In the rest of the paper, I will describe the CM processes that enhance development management and operations management.

8.3 **Continuous Maintenance (CM)**

As described in the previous sections, CI creates many artifacts, such as executables, meta-data and logs from code integration; results, data and logs from testing; warnings and reports from inspection. CI also depends on many repositories and artifacts, such as databases, storage and virtual machines. In addition, applications running in production also create and depend on many artifacts. Continuous maintenance (CM) consists of the processes that maintain these repositories and artifacts properly and consistently through automation, summarization, compaction, archival and removal.

CM aims to improve productivity, sustainability, and efficiency through automation. CM seeks to improve consistence and continuity within an IT department. The goal of this paper is to make companies aware of the CM processes, which complement and complete the continuous practices. Not all processes are applicable to all applications. CM processes can be classified into two phases: pre-production and post-production.

8.3.1 Pre-Production CM

For Commercial Off-the-Shelf (COTS) applications, the pre-production phase includes all the IT processes before the applications are released to the public. For Web applications, the pre-production phase refers to the IT processes before the applications are deployed to production. The pre-production phase includes all CI practices described in Section 8.2.1.

Above and beyond, the pre-production phase also includes processes such as feature verification, bug fixing, user acceptance testing, and staging. CM is responsible for maintaining the long term wellness of repositories and IT artifacts that support the pre-production phase. Some of the pre-production CM processes are listed below.

1) Repository Archival: Repositories are the core of many CI processes, such as automated build and testing. There are many version control repository software, such as *Concurrent Versions System* (CVS) [140] and Git [72], Perforce [141], and *Team Foundation Server* (TFS) [142]. Each software has its advantages and disadvantages, and organizes data in its own way. For example, Git accumulates data continuously. Without maintenance, the size of Git repositories will keep growing. In large companies, ever growing repositories may cause performance and management issues. CM monitors the content, size, and performance of the repositories, and keeps check of the relevant content. Older file versions that are no longer relevant to the product development can be archived. Nonetheless, these repositories often handle binary files poorly. For example, Git requires special tweaks to manage large binary files [143]. CM can automate theses corresponding processes.

CM can help maintain application branching policy. Such a policy enable pruning of branches that are no longer needed. According to the policy, CM can archive the branches that are abandoned or no longer relevant to the product line. In some cases, it may be worthwhile to apply some forms of data warehousing to dead branches, especially those with many binary materials and assets.

CM automatically archives content regularly. Archival does not imply removing artifacts forever. It may just be marking artifacts as historical and no longer relevant to the product development. CM is particularly important for companies using multiple types of repositories to support different technologies. CM will ensure that all types of repositories are maintained consistently.

2) Storage Cleanup: Besides repositories, many CI practices consume a lot of storage. For example, the continuous code integration process builds applications from repositories regularly and executes tests. For large applications, a complete build may take hours. When continuous code integration fails, the build's artifacts should be retained for investigation, while avoid filling up storage. Even if all tests pass, the IT department may retain the executables for additional assessment, such as performance and energy consumption. On top of the executables, the following details about each build should be retained:

- List of source code files used and their versions
- List of configuration files used and their versions
- List of binary files used and their versions
- Build and application execution log

If any tests fail, programmers need to know the exact conditions in which the tests fail. These additional test artifacts should be retained:

- Test logs and test results
- Databases and data used in the tests
- Virtual machines and containers

These artifacts are also retained from user acceptance test and staging, where written failure reports are provided to programmers after testing.

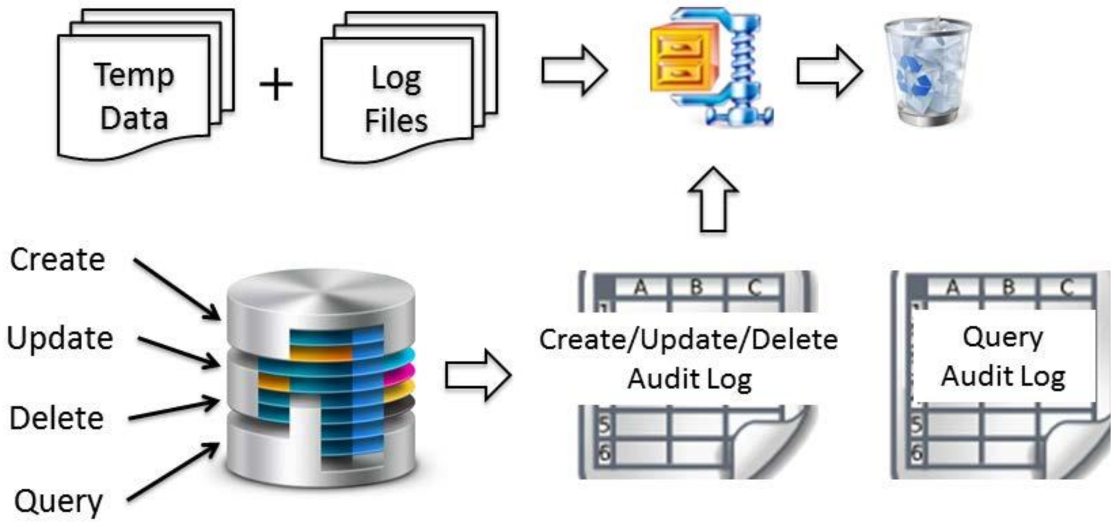


Figure 8.1 Temporary Data Cleanup and Logs Archival

CM manages metadata and retains relevant artifacts. When needed, CM can help recreate any executables from the repository according to the metadata. Retaining build and test artifacts devour a lot of storage. CM can deduplicate, compress, and/or archive large files to use less storage. CM can also support the business process that resolves integration problems. Once a build has served its business purpose, CM can eliminate irrelevant artifacts.

3) Management Systems Automation: Continuous code integration identifies conflicts; continuous testing identifies bugs; continuous inspection identifies quality issues. Companies may use different management systems (e.g. bug tracking system, task tracker, quality management system) to manage these affairs. Since the CI processes are being invoked continuously without human's involvement, CM should filter out redundant information for project stakeholders.

CM can simplify and enhance the management processes. For example, when automated tests fail, CM can use bug-dedupers and bug-clusterers [144] to determine whether the errors have been reported in the bug tracking system. If not, CM will automatically initialize bug reports on behalf of the programmers. CM can also employ bug repair, crash-dedupers, and crash-clusterers [145] to propose solutions for the programmers. CM can also close duplicated bugs.

Moreover, CM can utilize search based software engineering [146] to generate additional unit tests, which may reveal errors. In the presence of errors, search based software engineering can mutate and search possible programs for mutations that repair the errors. CM can bring these potential errors and solutions, not identified by continuous inspection, to the attention of the programmers.

In the pre-production phase, CM maintains a healthy development environment by managing artifacts used and generated during the CI processes. CM ensures all development artifacts are in good condition, so that programmers can concentrate on their source code.

8.3.2 Post-Production CM

For COTS applications, the post-production phase refers to the time after the applications are installed, configured, and have started serving users. For web applications, the post-production phase refers to the time after the applications are deployed to the production environments and are accessible by users.

Maintenance is necessary to keep applications sustainable. This section describes the automated CM processes that maintain applications in the post-production phase, involving both programmers and operational analysts.

1) Temporary Data Cleanup: Many applications create temporary data or files. The temporary data may be created intentionally, or left dangling because of abnormal termination. Regardless, programmers should know where temporary data may exist. Programmers need to work with operational analysts to identify potential temporary data. Then CM can automatically monitor, summarize, archive, and clean up temporary data, as shown in Figure 8.1.

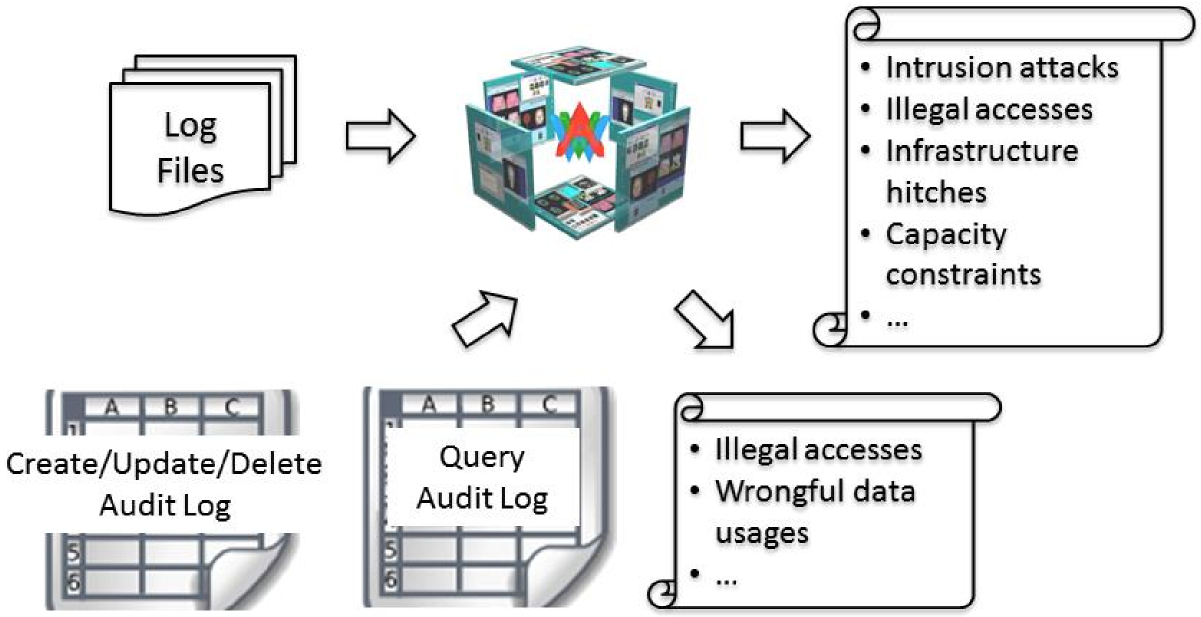


Figure 8.2 Exception or Violation Analysis

2) Logs Archival: Many applications keep execution logs about connection requests, service requests, executed processes, exceptions, and errors. Under normal situation, the logs will not be examined. When problems occur, the logs will be inspected. CM can analyze the logs and store the relevant statistic data, then compress and archive the logs for a period of time before eliminating them, as shown in Figure 8.1.

Many applications also keep audit logs. Audit logs contain metadata about data *creation, retrieval, update, and deletion* (CRUD). CM can analyze and summarize the audit logs as benchmark. Audit logs can grow rapidly, especially the data retrieval audit logs. Therefore, CM should monitor the size of the audit logs, then compress and archive the logs as needed. Audit logs are very important for many businesses. CM should keep the archived audit logs easily accessible for inquiry or analysis.

3) Exception or Violation Analysis: Execution logs and audit logs contain valuable information. CM can analyze them before archiving.

Analyzing execution logs can identify potential problems such as intrusion attacks, illegal accesses, infrastructure hitches, and capacity constraints. CM can bring these potential problems to the responsible personnel. Defining and continuously enhancing the exception analyzing rules can prevent potential problems in the future, as shown in Figure 8.2.

Analyzing audit logs can identify wrongful data usages. For example, there were incidents where health service providers had been illegally browsing patients' medical records [147] [148]. CM could have identified the wrongful behavior through analyzing the retrieval audit logs. Subject matter experts may be involved in defining the rules for audit logs analysis. The rules should be reviewed and enhanced continuously.

4) Data Elimination: In some businesses, there are legal obligations to eliminate obsolete data. For example, the law enforcement systems may need to remove juveniles' records once the offenders become adults. E-commercial systems may need to purge cardholder data after retention period [149]. Subject matter experts and legal advisors may involve in defining the data elimination rules for CM to remove obsolete data for legal compliance.

5) Data Warehousing and Analytics: Proper data warehousing and analytic improve business value, and speed time to insight [150]. Sometimes production data is too complicated and sensitive for data warehousing. CM can employ the *Extract, Transform and Load* (ETL) tools provided by the *database management systems* (DBMSs) to periodically extract analyzable data from production databases, then filter, convert, mask, and replace sensitive data before loading data into data warehouses for analysis. Subject matter experts and data analysts may need to review the data conversion rules to make sure sensitive information cannot be mined from the data warehouses.

In the post-production phase, CM maintains applications' sustainability by managing operational data and logs. So that operational analysts can focus on handling exceptions, and will be more aware about potential problems.

8.4 **Conclusion**

Many artifacts, such as executable, meta-data, test data, logs, reports, virtual machines, and containers, are used and generated by the continuous integration processes during development. In addition, data and logs are generated by applications running in production. Managing repositories and these artifacts is onerous and not adequately addressed in the continuous literature. Thus ***continuous maintenance*** is proposed to maintain these artifacts and extract values out of them through analysis, summarization, compaction, archival, and removal. This paper identifies specific automatable maintenance processes needed before production in development, and after production in operation that are benefited by continuous maintenance.

Continuous maintenance attempts to automate many manual maintenance processes to reduce maintenance costs. This allows companies who deploy continuous maintenance to allocate a bigger portion of their IT budget to innovation [151], hence increase return on investment.

Since this is a relatively new topic, there are many research questions to be answered:

- How many companies have implemented what CM processes?
- How do companies deploy the CM processes?
- Who is accountable for the CM processes?
- Have companies implemented or purchased CM tools?
- Where do programmers and operational analysts see the needs of cooperation for maintenance?
- What are the challenges in their cooperation?
- What other CM processes exist?
- Can CM policy, guideline or checklist be defined?

9 Single Source of Truth (SSOT) for Service Oriented Architecture (SOA)

In Section 2.2.1, I described survey results which concluded that DevOps improves IT performance. DevOps improves IT performance by creating a pipeline to deploy application as fast as possible.

Running the whole pipeline for a huge monolithic application is often slow. Same applies to testing, packaging, and deployment. On the other hand, microservices are much faster for the simple reason that they are far smaller (p.23) [53].

Microservices are slowly becoming the preferred way to build big, easy to maintain and highly scalable systems (p.9) [53].

Single Source of Truth (SOA) is a software design pattern that packages functionality as a suite of interoperable services to be shared by multiple separated systems. Richards called Microservices the "incarnation of SOA" [36]. Each microservice should be a Single Source of Truth (SSOT) within the system. While most publications focus on the technical and architectural perspectives of microservices, no publication examines the relationship and interaction between microservices. I filled the gap by writing the paper "Single Source of Truth (SSOT) for Service Oriented Architecture (SOA)" [37], which depicted the relationship and interaction between services in SOA, hence microservices.

The paper was published in the Proceeding of 2014 International Conference on Service-Oriented Computing (ICSOC). The final publication is available at Springer via http://dx.doi.org/10.1007/978-3-662-45391-9_50. The paper is reprinted in this section, with permission from Candy Pang and Duane Szafron.

The major contribution of this paper is defined two types of SSOT relationships (mutable and immutable), their operations, interactions and benefits.

The benefits of the mutable SSOT include the following:

- No need to keep foreign data on local resource.
- No need to manage foreign data changes in local database.
- No need to store a large amount of foreign data, when only a small amount is used.
- Always get the latest up-to-date correct data.

In addition to the benefits of mutable SSOT, immutable SSOT provide these additional benefits:

- In large enterprises or government agencies, SSOT allows data to be divided into separate units, therefore enhance maintainability, reusability, scalability and accountability.
- Enhanced data quality, consistency and protection.
- Provides only limited data to the users according to users' authorization, instead of exposing the whole data set.

Base on the defined SSOT operations, development tool can easily be created to generate SSOT service code with corresponding client access code, which makes microservices and DevOps adoption easier.

For this publication, I was responsible for the identification and design of the research program, the collection of data, the construction of apparatus, the performance of experiments, the analysis of the research data, and the manuscript composition. Dr. Duane Szafron was the supervisory author, and was involved with concept formation and manuscript composition.

Impact: As of writing, SSOT for SOA has been cited one time according to the Google Scholar [85], which is a Doctor of Philosophy thesis that studied user interface design [152].

Abstract

Enterprises have embraced Service Oriented Architecture (SOA) for years. With SOA, each business entity should be the Single Source of Truth (SSOT) of its data, and offer data services to other entities. Instead of sharing data through services, many business entities still share data through data replication. Replicating data causes inconsistencies and interoperability challenges. Even when there is a single authoritative source, that resolves inconsistencies, the data copies may end up being out-of-sync and cause errors. This paper describes how to use a SSOT service to eliminate data replication, enforce data autonomy, advocate data self-containment, and enhance data maintenance. Both mutable and immutable SSOT relationships (mappings) are considered. This paper describes the challenges, solutions, interactions and abstractions between the SSOT data service providers and the loosely coupled data consumers. It also assesses the performance and future usage of a SSOT service.

Keywords: Single Source of Truth, Service Composition, Service Oriented Architecture (SOA), Software Design Concept, Software Engineering.

9.1 Introduction

Before embracing the Service Oriented Architecture (SOA), enterprises or business entities used to share data through data replication. Replicating data across multiple systems gives rise to inconsistencies and interoperability challenges. In some cases, one of the systems is treated as the "authoritative" source or the Single Source of Truth (SSOT). The authoritative source's data is replicated to the clients' databases, resulting in data layer synchronization challenges. Independent client's data transformation or modification can result in data discrepancies that require manual intervention. A better solution is to hide the data layer from the clients in the SOA.

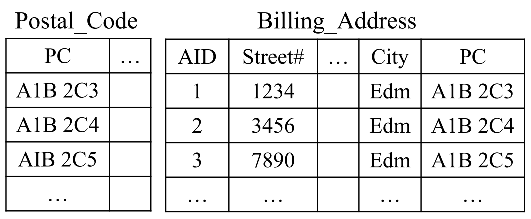
In the SOA, a SSOT should associate with clients through the service layer, instead of the data layer. This would alleviate data replication and the related problems. The authoritative source identified as the SSOT should provide data services for the clients. In this approach, clients maintain mappings to the SSOT data, but do not replicate the SSOT data. Unfortunately, many enterprises have yet to overhaul data layer replication into a SSOT service. Lack of experience in SSOT service implementation may have hindered enterprises from the migration.

This paper describes a SSOT service model for two common data sharing scenarios: mutable and immutable data sources. We use two motivating examples to illustrate the variants: (a) the management of Postal Codes (PC) and (b) the management of Electronic Patient Records (EPR). The proposed SSOT service model is useful for any business entity that maintains full ownership of its data, and does not want the clients to duplicate its data, but allows data access by restricted queries. The value of the model will be illustrated by the PC and EPR examples.

Most business applications use addresses. In Canada, Canada Post is the single authoritative agency that manages PCs for mail-delivery addresses. Each address should have exactly one PC, while each PC covers an area with multiple addresses. Most business applications collect address information from their customers, and store customers' addresses with PCs in their local databases. Periodically, business applications also replicate Canada Post's PCs to their local databases for data-validation purpose. For example, an application using billing addresses, which require PCs, may have a Billing_Address table and a Postal_Code table as shown in Figure 9.1(a). The Postal_Code table contains all valid PCs periodically replicated

from Canada Post. Before adding a new address to the Billing_Address table, the application checks the validity of the provided PC, i.e., whether the PC exists in the Postal_Code table. If so, the application will add the new address to the Billing_Address table. This process can only validate the existence of the PC, but it cannot validate whether the PC is correct for that address. There is a mapping between the PC and the billing address.
A mapping that changes over time is called a mutable mapping, and a mapping that does not change is immutable. Canada Post changes PCs from time to time. PCs can be inserted, updated, deleted, split or merged. The application needs to synchronize the Postal_Code table with Canada Post, and, if necessary, to correct the PCs in the Billing_Address table. Since the PC that is mapped to a billing address can change over time, the mapping between PC and billing address is an example of a mutable mapping. Mutable mappings are often subject to synchronization errors. For example, Canada Post only provides PC changes to subscribers monthly. Therefore, the subscribers' Postal_Code tables are out-of-sync with Canada Post most of the time. As shown in Figure 9.1(b), when a PC is updated (from A1B 2C3 to A2B 2C3), the corresponding mappings in the Billing_Address table can be updated. However, when a PC is deleted (from A1B 2C4 to none) or split (from A1B 2C5 to A1B 2C5 and AIB 2C8), there is no simple solution to correct the mappings in the Billing_Address table, by using the updated PC list.

As a second example, let us consider a regional Electronic Patient Record (EPR) system that manages patients' health numbers (PHNs), names and contacts. Each patient in the region receives services from multiple healthcare providers, with different specialties. Each healthcare provider obtains patient information directly from the patient during the patient's visit, containing information included in the EPR. Then, each provider independently stores the patient's information in its local database. In this model, a provider-specific patient record may be inconsistent with the regional EPR. In principle, each provider-specific patient record could be mapped to a corresponding EPR in the regional authoritative system. In this case, the mapping between an EPR and a provider-specific record is immutable. The mapping is immutable despite the fact that patient's data may change. For example, when a patient changes his/her name, the patient's EPR will be updated, but the patient's EPR is still mapped to the same provider-specific record. Therefore, the mapping is immutable.



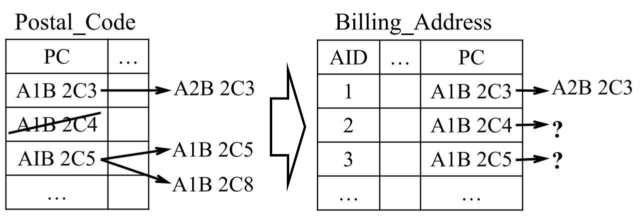
Postal_Code

PC	…
A1B 2C3	
A1B 2C4	
AIB 2C5	
…	

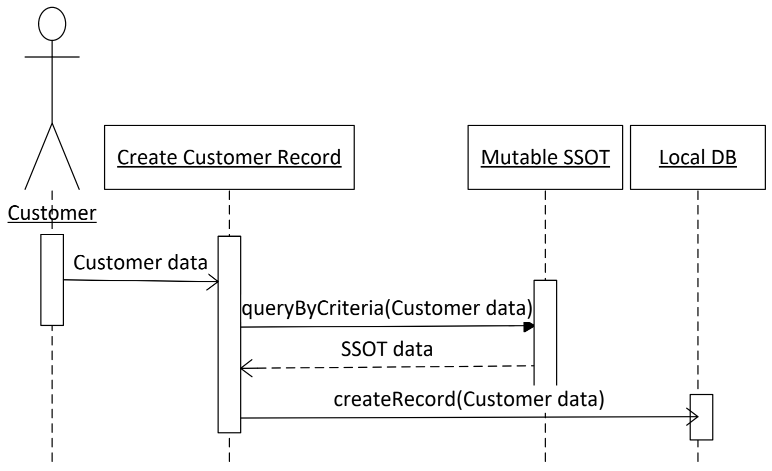
Billing_Address

AID	Street#	…	City	PC
1	1234		Edm	A1B 2C3
2	3456		Edm	A1B 2C4
3	7890		Edm	A1B 2C5
…	…		…	…

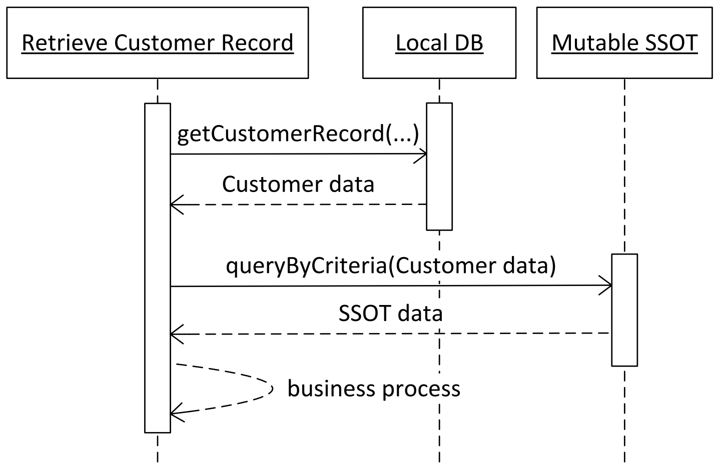
(a) Billing address tables



(b) PCs update, delete and split



(c) Create record with mutable SSOT



(d) Retrieve record with mutable SSOT

Figure 9.1 The Postal Codes Use Case

The SOA paradigm can alleviate data synchronization issues. Clients using data that already exists in an authoritative source will not replicate the authoritative data in their local databases. Instead, the authoritative source acts as a SSOT service that provides clients the authoritative data. In the examples above, Canada Post serves as the SSOT service for PCs and a regional health authority provides the SSOT service for EPRs. Clients access PCs and EPRs by invoking SSOT services, without replicating SSOT data in their local databases. Therefore, clients do not need to manage and synchronize data with the SSOT. The SSOT can also shield its autonomy from the clients. We advocate the SSOT service model over data-replication. This paper is structured as follows. Section 9.2 and 9.3 illustrate the challenges and solutions associated with the mutable and the immutable SSOT by the PC and the EPR use cases respectively. Section 9.4 evaluates the performance of the SSOT service. Section 9.5 describes the related works. Section 9.6 recommends future works, and Section 9.7 concludes the paper by enumerating SSOT's benefits.

9.2 **Mutable SSOT Service**

A SSOT service that manages mutable mappings is called a mutable SSOT service. In this case, the mapping between a SSOT record (each individual PC in our example) and a client application record (each billing address in our example) can change over time. Clients need to invoke the mutable SSOT service with query criteria. Therefore, a mutable SSOT service supports at least the query-by-criteria operation.

For example, in the PC use case, Canada Post maintains a mutable PC SSOT service that provides a single web service operation, PC-query. The PC-query operation takes PC query criteria as input. Clients may specify Street Number, Number Suffix, Unit/Suite Apartment, Street Name, Street Type, Street Direction, City, and/or Province as criteria. The PC-query operation returns a set of PCs that match the criteria.

This PC SSOT service can replace data replication. For example, in Figure 9.1(a), the PC column in the Billing_Address table and the Postal_Code table are replicated data that can be dropped in favor of invoking the PC-query operation provided by the mutable PC SSOT service. The PC query criteria will come from the remaining columns (e.g. Street#, City) in the Billing_Address table. When the application needs the PC of a billing address, it will use the address data in the Billing_Address table as query criteria to retrieve the PC from the PC SSOT.

The rest of this section will use the PC use case to illustrate how clients can use a mutable SSOT service to replace data replication.

9.2.1 Client-Record Creation

Figure 9.1 (c) depicts how an application can use mutable SSOT service for data validation during record creation. In the PC use case, when the application receives a new billing address from a customer, the application queries the PC SSOT using the customer address. If the PC SSOT returns a single valid PC, then the address is valid. The application proceeds to create a new Billing_Address record for the customer.

If the PC SSOT returns more than one PC, then the customer address is not definitive. For example, if the customer address contains only the city field, then the PC SSOT will return all the PCs for the city. In principle, the application should not accept a non-definitive address as a billing address. Therefore, the application would seek additional address details from the customer. Similarly, if the PC SSOT returns no PC for the customer address, the application should alert the customer that the address is invalid and request the user to take remedial action.

In contrast, the data-replication model may allow non-definitive or invalid billing addresses in the application's database. Using a mutable SSOT service not only eliminates data replication, but also enhances data quality.

Different clients may have different processes for the SSOT response. The application in the PC use case expects a single valid PC from the response. Other applications may iterate the response records to select the most desired result. To support different clients' processes, the mutable SSOT query-by-criteria operation may return additional information. For example, the PC-query operation may return other address fields (Street Number, Unit/Suite Apartment, Street Name, etc.) in addition to the PC. Clients can use the additional address fields to filter the response records.

9.2.2 Client-Record Retrieval

Since the SSOT data is excluded from the clients' local databases, each client needs to combine its local data with the SSOT data to compose the complete data records. The client data retrieval process is depicted in Figure 9.1(d).

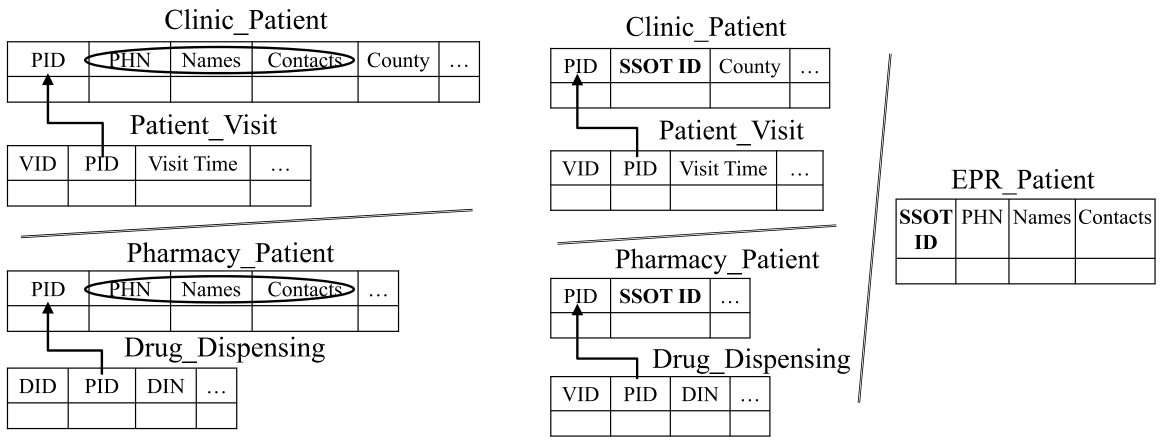
In the PC use case, the application first retrieves a Billing_Address record from its local database, then uses the local address data as query criteria to invoke the PCquery operation, and retrieve the up-to-date PC from the PC SSOT. If the PC of the billing address has changed since the last retrieval, then the PC SSOT will return a different PC from the last retrieval, but it will be the correct PC.

9.3 Immutable SSOT Service

A SSOT service that manages an immutable mapping is called an immutable SSOT service. The mapping cannot change over time. Each record in the client application has a permanent static relationship with a SSOT record. To establish this mapping, a client record is associated with a SSOT record using a unique permanent single source of truth identifier (SSOT-ID). A SSOT-ID is conceptually equivalent to the resource identifier in the Resource Description Framework (RDF) [153] and the unique identifier in the Representational State Transfer (REST) [154] architecture. Each record represents a resource in the SSOT. We will use the term resource instead of record since each SSOT resource may contain multiple child-records. For example, an EPR may contain multiple names, addresses and phone numbers as child-records. Each resource has a unique permanent identifier called the SSOT-ID. Clients use the query-by-SSOT-ID operation to retrieve resource details, which include the child-records.

The rest of this section will illustrate the immutable SSOT service through the EPR use case. Assume that a clinic application and a pharmacy application both need patients' health numbers (PHNs), names and contacts, along with their own provider specific data. Traditionally, in the data-replication model, the clinic application would have a Clinic_Patient table and a Patient_Visit table, and the pharmacy application would have a Pharmacy_Patient table and a Drug_Dispensing table, as shown in Figure 9.2(a). The clinic application assigns a county to each patient using the patient's home address, while the pharmacy application does not. In the data-replication model, the PHNs, names and contacts located in the clinic's and pharmacy's databases may have errors or be inconsistent with the authoritative EPR system. When patients move, patients must notify the EPR authority, the clinic and the pharmacy individually.

Actually, PHNs, names and contacts are readily available in the regional EPR system. An immutable EPR SSOT service can eliminate data replication from the clinic and the pharmacy databases. After eliminating the replicated EPR data, Figure 9.2(b) shows the new clinic application and pharmacy application tables. In the new Clinic_Patient and Pharmacy_Patient tables, the EPR columns are replaced by the SSOT-ID column. Unlike the PC use case, immutable SSOT clients do not routinely use the query-by-criteria operation to obtain SSOT data. Instead, clients can invoke the query-by-SSOTID operation to retrieve SSOT data from the immutable SSOT service.



(a) Providers' table in replication model **(b) Providers' table in immutable SSOT model**

EPR SSOT Query Criteria:
- Lastname = 'PANG%'
- Phone#='7807654322'

EPR SSOT Query Response Object:
- SsotID = 'DYXMSFK9SJ2'
- Lastname = 'PANG'
- Firstname = 'CANDY'
- Phone# = '7807654321'
- Health# = '####5-3623'
- Address = '4523 *****'
- Birthday = 'MM-09-YYYY'

(c) Partial response example

Figure 9.2 The Electronic Health Record Use Case I

Each immutable SSOT service should provide at least four operations: (a) query-by-criteria, (b) query-by-SSOT-ID, (c) update subscription, and (d) deletion subscription. The potential risks of using the immutable SSOT service query-by-criteria operation are illustrated in Section 9.3.1. The rest of the sub-sections describe different usages of the immutable SSOT operations.

9.3.1 Query-by-Criteria

An immutable SSOT service could provide the same query-by-criteria operation as a mutable SSOT service. Clients could query the SSOT service by criteria and receive a correlated set of results matching the criteria. However, in some situations, the immutable SSOT query-by-criteria operation may sustain a privacy violation risk. In the EPR use case, if the query-by-criteria operation returns all resources matching any given set of criteria, then any client can browse the

EPR data, which violate patients' privacy. For example, a client may specify Firstname='JOHN' as the query criteria for the EPR SSOT query-by-criteria operation. In response, the EPR SSOT returns all patients with first name equals 'JOHN'. The large response set may violate many patients' privacy, since many of the set may not be patients of the querying facility.

To avoid browsing, the query-by-criteria operation could specify a maximum number of returned records (i.e. query-limit). If the response set is larger than the query-limit, the service would return an error. At which point the client needs to refine the query criteria and queries again.

A safer query-by-criteria operation may also demand more than one query criterion to avoid brute force hacking. For example, the EPR SSOT query-by-criteria operation could reject single criterion queries to avoid PHN cracking by querying with randomly generated PHN's until a valid resource is returned.

For further privacy protection, the query-by-criteria response set can be filtered to contain partial data. The partial data must include the complete SSOT-ID and sufficient information to identify a SSOT resource. Figure 9.2(c) shows a set of query criteria for the EPR SSOT query-by-criteria operation, and one partial response record. The partial record includes the SSOT-ID and only part of the PHN, address and birthday. Using the partial record, a client should be able to determine whether the EPR maps to the targeted patient. Once a partial record is selected, the client can use the SSOTID to retrieve the resource details using the query-by-SSOT-ID operation.

9.3.2 Query-by-SSOT-ID

If the query-by-criteria operation returns only partial data for privacy protection, the immutable SSOT service must provide a query-by-SSOT-ID operation, which takes a SSOT-ID as input. In response to a query-by-SSOT-ID request, the immutable SSOT provides details of the resource corresponding to the provided SSOT-ID, but only the details that the client is authorized to see. This allows the SSOT service to distinguish between client access permissions, providing different information to different facilities, such as pharmacies, clinics and acute-care facilities.

To protect data privacy, the immutable SSOT should include a proper auditing mechanism to detect, identify and stop improper browsing behavior or unlawful use of data. In the EPR use case, if a client continually invokes the EPR SSOT query-by-SSOT-ID operation with randomly generated or guessed SSOT-IDs, the EPR SSOT auditing mechanism should detect and deter the client.

Each SSOT-ID is effectively a foreign key to a remote SSOT resource. Since the SSOT and clients are loosely coupled, the foreign key constraints in the client data cannot be enforced at the SSOT. An alternate foreign key constraint handling mechanism will be discussed in the later sub-sections.

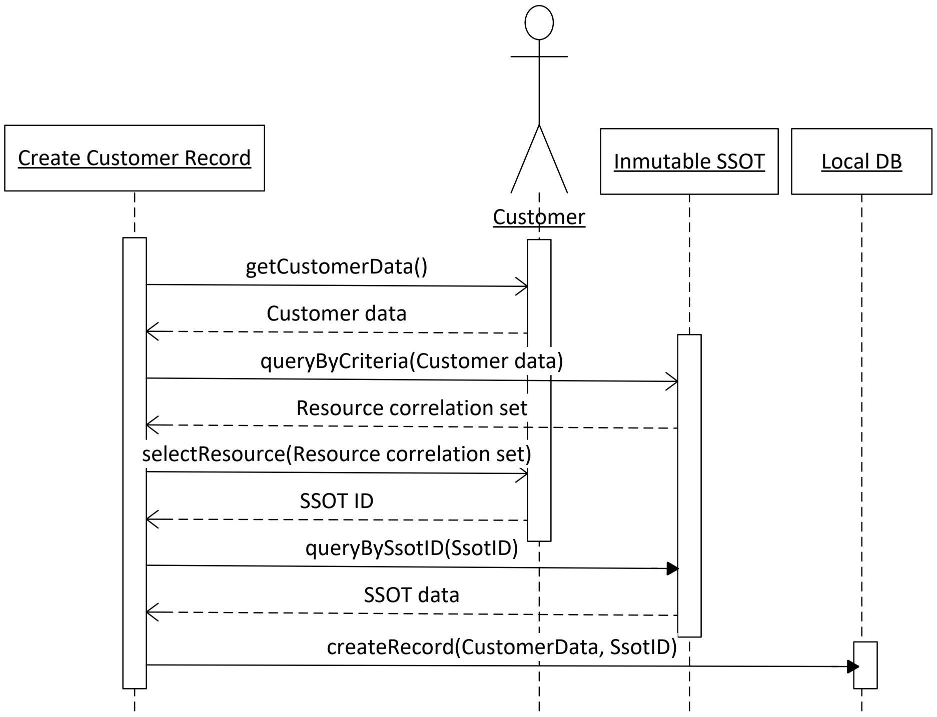
9.3.3 Client-record Creation

Figure 9.3 (d) depicts the role of an immutable SSOT service during client record creation. Using the clinic application in the EPR use case as an illustration, when a patient first visits the clinic, the clinic application needs the patient's SSOT-ID. The patient provides personal data to the clinic. The clinic application invokes the EPR SSOT query-by-criteria operation with the patient's data. From the returned set of partial EPRs, the clinic and patient together identify the correct EPR. With the SSOT-ID from the selected partial EPR, the clinic application invokes the query-by-SSOT-ID operation to retrieve the portion of the patient's EPR that is permitted to the clinic. The clinic application then assigns a county to the patient according to the patient's home address in the EPR. With the patient's SSOT-ID and assigned county, the clinic application adds a new record to the Clinic_Patient table for the patient. Similarly, when a patient first visits the pharmacy, the pharmacy application uses the EPR SSOT query-by-criteria operation to retrieve a partial EPR of the patient. The pharmacy application gets the patient's SSOT-ID from the partial EPR. Since the pharmacy data does not depend on data in the EPR, the pharmacy application can add a new record to the Pharmacy_Patient table for the patient with the patient's SSOT-ID.

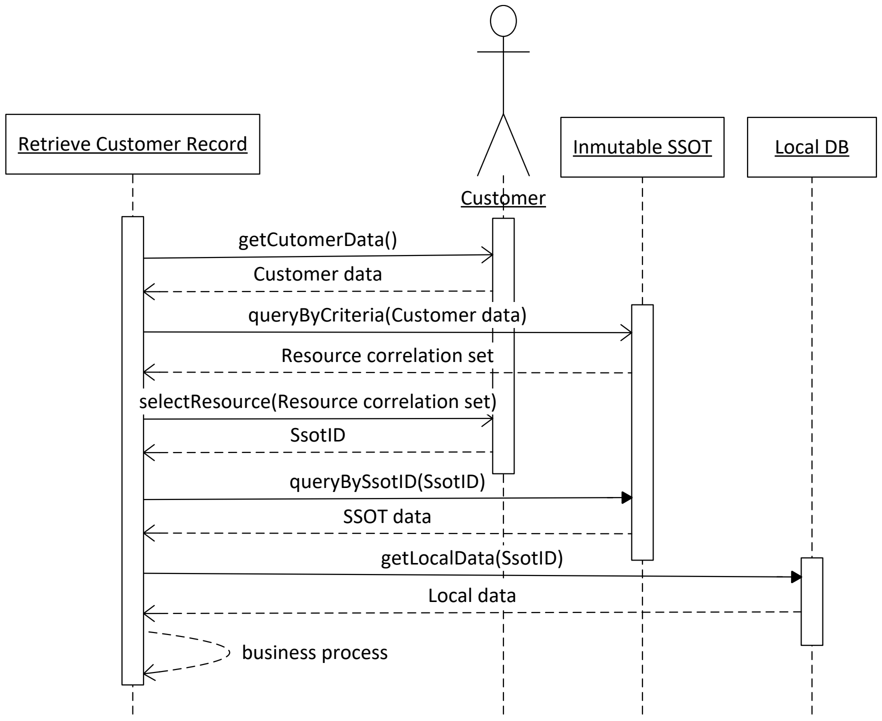
9.3.4 Client-record Retrieval

As with the mutable SSOT service, the immutable SSOT clients need to combine the SSOT data with the local data to compose complete data records. The data retrieval process is depicted in Figure 9.3(e).

In the EPR use case, when a patient revisits the clinic or the pharmacy, the clinic and pharmacy applications use the EPR SSOT query-by-criteria operation to retrieve the patient's SSOT-ID. With the SSOT-ID, the applications fetch the permission filtered EPR with query-by-SSOT-ID. Using the SSOT-ID again, the applications retrieve patient's local data from the local databases, i.e. the Clinic_Patient, Patient_Visit, Pharmacy_Patient and Drug_Dispensing tables in Figure 9.2(b). Finally, the applications combine the EPR and local data to instantiate a complete patient record.

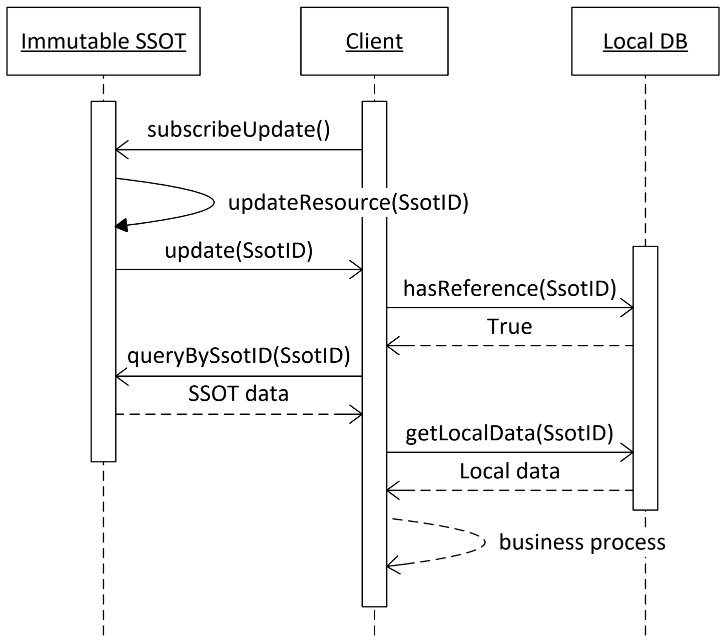


(d) Create record with immutable SSOT



(e) Retrieve record with immutable SSOT

Figure 9.3 The Electronic Health Record Use Case II



(f) Immutable SSOT update subscription

Figure 9.4 The Electronic Health Record Use Case III

9.3.5 Constructing SSOT-IDs

Since the SSOT and the clients are loosely coupled, the clients rely on the SSOT-IDs to be unique and permanent. Therefore, it is important to select a proper data type, length, format and representation for the SSOT-ID. For example, the Canadian Social Insurance Number (SIN) has 9 digits. The first digit of the SIN represents the owner's residential status. Similarly, a SSOT-ID can have embedded representations. The design of the SSOT-ID deserves extraordinary attention to ensure uniqueness and permanency. For example, the SIN is unique, but not permanent. When an owner's residential status changes, a new SIN may be assigned. Therefore, SIN is not a good SSOT-ID candidate. In addition, public data items are not good SSOT-ID candidates.

An SSOT-ID does not need to be a single value. It can also be a composite value, as long as it is unique and permanent. Part of the SSOT-ID can be a fixed-length sequential number, to ensure its uniqueness. Since the SSOT-ID will be shared between systems, it is recommended that the SSOT-ID take a different format from the SSOT database standard, so that the SSOT database standard will not be exposed.

Clients store the SSOT-IDs in their local databases. The SSOT-IDs are guaranteed to be unique and permanent. Therefore, clients may consider using the SSOT-IDs as the primary keys in their local databases. If the SSOT-ID data type and size do not match the local database standard, we recommend that the local database create its own local primary key, and use the SSOT-ID as a foreign key. In the EPR use case, the Clinic_Patient table could have used the EPR SSOT-ID as the primary key. Since the primary key of the Clinic_Patient table will be a foreign key of the Patient_Visit table, the clinic application created its local primary key (PID in the Clinic_Patient table), to preserve data type consistency between tables.

The clients should also consider whether the sequence of the primary keys matter to the local business logic. In the EPR use case, it is likely that a portion of the EPR SSOT-ID is a sequential number. The clinic serves only a relatively small number of patients in the regional EPR system. Therefore, only a small number of the EPR SSOT-IDs will be imported into the clinic's database. If the clinic application uses the EPR SSOT-IDs as its primary key, then the primary key will have a lot of gaps in its sequence. In addition, the order of the primary keys will not represent the order in which patients' records are added to the clinic's database.

9.3.6 Update Subscription

Data updates for an immutable SSOT may affect clients. In the EPR use case, the clinic application assigns a patient's county based on a patient's home address. When a patient moves, the clinic application may assign a different county for the patient. In this case, data updates in the EPR SSOT affect the clinic application. On the other hand, none of the pharmacy application local data depends on the EPR SSOT data. Therefore, data updates in the EPR SSOT do not affect the pharmacy application.

The SSOT cannot determine how data updates will affect the loosely coupled clients. Clients are responsible for managing their own data. Therefore, the immutable SSOT must provide an update subscription operation. If a client is concerned about data updates in the SSOT, then the client is responsible for subscribing to the SSOT update service through the update subscription operation.

After a SSOT resource is updated, the SSOT will send an update message with the SSOT-ID of the updated resource to the subscribers. When the subscriber receives the update message, the subscriber can check whether the SSOT-ID is referenced locally. If not, the subscriber can ignore the update message. If the SSOT-ID is referenced locally, then the subscriber can fetch the resource details using the query-by-SSOT-ID operation. Based on the latest resource details, the subscriber may update its local data accordingly. The update subscription process is depicted in Figure 9.4(f).

In the EPR use case, the clinic application would subscribe to the EPR SSOT update service. When an EPR is updated, the clinic application will receive an update message with the SSOT-ID of the updated EPR. The clinic application determines whether the SSOT-ID is referenced locally. If so, it retrieves the patient details from the EPR SSOT using the query-by-SSOT-ID operation. Then the clinic application can determine whether the patient's latest home address matches the clinic-assigned county in the local database. If not, it updates the local database accordingly. On the other hand, the pharmacy application is not affected by EPR updates. Therefore, the pharmacy application would not subscribe to the EPR SSOT update service. Notice that the pharmacy still relies on the SSOT for the latest EPR patient information. However, it does not subscribe for updates since it does not need to update its own local database, when EPR data changes.

9.3.7 Deletion Subscription

Just like any other data, the SSOT data can be deleted. Since the SSOT is loosely coupled with its clients, clients cannot put a foreign key constraint on the SSOT to restrain the SSOT from deleting data. Instead, the SSOT provides a deletion service. When a resource is deleted from the SSOT, the SSOT will send a deletion message to the subscriber. Clients need to determine whether they should subscribe to the SSOT deletion service using the deletion subscription operation.

In the EPR use case, deleting a patient from the EPR SSOT may create broken links in the Clinic_Patient and Pharmacy_Patient tables. Therefore, the clinic and pharmacy applications should both subscribe to the EPR SSOT deletion service.

If the SSOT physically deletes the resource, then the deletion message should contain the last version of the resource before the deletion and the reason for deletion. Clients can use this information to determine how to handle the deleted data.

In the EPR use case, a patient may move from the region and be deleted from the EPR SSOT. The clinic and pharmacy applications will receive a deletion message from the EPR SSOT with the last version of the patient's EPR. The applications may store or ignore the EPR in the deletion message. Depending on the reason for deletion, the applications can delete, archive or mark the patient's local record inactive.

Alternatively, the SSOT may logically delete a resource. In this situation, the deletion message will only contain the resource SSOT-ID and the reason for deletion. The client can still fetch the logically deleted resource using the query-by-SSOT-ID operations. The query-by-SSOT-ID operation would return the corresponding resource but flagged as deleted. The query-by-criteria operation could exclude logically deleted resources from the response correlated set, or provide them and flag them as deleted.

In the EPR use case, if patients are only logically deleted from the EPR SSOT, the clinic and pharmacy applications may mark the patient inactive in their local databases. If EPR records are only logically deleted, then clients should always check the deleted flags on the EPR, since logically deleted records may later be undeleted.

9.3.8 Additional Operations

An SSOT creation subscription operation is not recommended. If clients can subscribe to SSOT data creation, then clients can replicate the whole SSOT database, which violates the purpose of using SSOT. Clients should only link to the SSOT resources related to their operational mandates, but not replicate the SSOT data.

In addition to the four mandatory operations, an immutable SSOT service might provide additional resource create, retrieve, update and delete (CRUD) operations. In the EPR use case, if a patient has not registered with the regional EPR SSOT, then the clinic and pharmacy applications would not find the patient through the query-by-criteria operation. If the EPR SSOT also provides a patient creation operation, then the authorized clinic or pharmacy personnel can create SSOT EPRs as needed.

If the clinic or pharmacy personnel are not authorized to create SSOT EPRs, then the un-registered patients need to register with the EPR authority later. In the meantime, the applications can create a local temporary file to store the patient's data. An optional column (TEMP FILE#) can be added to the Clinic_Patient and Pharmacy_Patient tables to keep track of the temporary file number. In the absence of the SSOT ID and existence of the TEMP FILE#, the applications will not query the EPR SSOT for patient's information, but retrieve data from the local temporary file. After the patient registers with the EPR SSOT, the applications can insert the EPR SSOTID into the Clinic_Patient and Pharmacy_Patient tables, and delete the temporary file. At this point, the application returns to normal processing.

9.4 Performance and Quality-of-Service (QoS)

Despite the data-synchronization challenges, the data-replication model has a performance advantage over the SSOT service model. In the SSOT service model, client applications make extra service invocations to the SSOT during record creation and retrieval, which may affect user experience. Therefore, we implemented experiments to evaluate how the extra SSOT service invocations might affect user wait times.

The experiments evaluated delay caused by the SSOT service invocations. They were conducted on the institution's network supporting ~40,000 students. The SSOT service was hosted on a workstation in a departmental subnet. In the experiments, client applications accessed the SSOT service through Wi-Fi on the institute's public network during regular office hours. This condition simulated an enterprise network supporting multiple sub-divisions.

The experiment results show that if the SSOT service and the client applications are located within the same enterprise network, then the service invocation costs less than 20 milliseconds per call. This indicates that users should not notice any performance deterioration. If the SSOT service is available across a wide area network, the performance primarily depends on the transmission delay between the public SSOT service and the clients. Clients should benchmark the transmission delay to determine the actual performance effect.

In dynamic web service composition [155], clients select web service providers according to their published quality-of-service (QoS) [156]. Even though, SSOT's clients will likely access the SSOT service statically, the SSOT service should publish the following QoS metrics per operation:

- **Operational hours:** the regular SSOT servicing hours.
- **Maintenance schedule:** the changes and release schedule.
- **Reliability:** the SSOT's ability to perform the operation without errors.
- **Request-processing time:** the maximum and average time required for the SSOT service to complete the operations.

The SSOT may publish additional QoS metrics, such as capacity, performance, robustness, accuracy and more [155]. Clients can design their usage of the SSOT service according to the SSOT's QoS metrics.

If the SSOT service is part of a larger enterprise or jurisdiction, then the SSOT service usually has the same operational hours and maintenance schedules as their clients. Public SSOT services, like Canada Post, are usually available 24x7.

9.5 Related Work

Most SSOT is implemented on the data layer. Ives *et al.* [157] suggest synchronizing distributed data on the data layer. Ives *et al.* propose a "Collaborative Data Sharing System (CDSS) [that] models the exchange of data among sites as update propagation among peers, which is subject to transformation (schema mapping), filtering (based on policies about source authority), and local revision or replacement of data." Since data across multiple sites are continuously "updated, cleaned and annotated", cross-site synchronization has to deal with issues such as data correctness, schema and terminology consistence, and timing. These data layer synchronization hurdles highlight the advantage of our SSOT service model that eliminates data layer synchronization.

Others try to implement SSOT using an Enterprise Service Bus (ESB) [158], in which a SSOT is defined. Clients duplicate the SSOT data locally, and subscribe to ESB for SSOT updates. Whenever the SSOT is updated, clients synchronize with the SSOT by repeating the changes in their local copies. Our SSOT model totally avoids data duplication at the clients' site.

Instead of a data-centric model for SSOT, some research has turned to artifact-centric modeling [159]. An artifact is a set of name-value-pairs related to a business process or task, where data represents business objects. In the artifact-centric model, each artifact instance is shared between all process participants. The participants get information from the artifact and change the state of the artifact to accomplish the process goal. Since the artifacts are shared between process participants, access and transaction control is necessary. Hull [160] suggests using artifact-centric hubs to facilitate communication and synchronization between the participants. Our SSOT service model does not require complicated facilitation or a centralized hub.

Other researchers have proposed the Personal Information Management (PIM) [161] model. In our model, the SSOT does not have any knowledge about its clients' data. However, the PIM model finds, links, groups and manages clients' data references to the source. PIM is a centralized data management model, while SSOT is a distributed data management model.

Finally, Ludwig *et al.* [162] propose a decentralized approach to manage distributed service configurations. The proposed solution uses RESTful services to exchange configuration data between hosts, and a subscription mechanism to manage changes. This approach endorses a data perspective similar to an SSOT service, where each data source maintains self-contained autonomous data. Data is not synchronized across multiple sites. Sources and clients are statically bound.

9.6 **Future Work**

A mutable SSOT service has one standard operation: query-by-criteria. An immutable SSOT service has four standard operations: query-by-criteria, query-by-SSOT-ID, update subscription, and deletion subscription. Based on these standard operations, we defined Web Service Definition Language (WSDL) extensions for the mutable and immutable SSOT services with corresponding templates. Because of the limited space, we do not include the SSOT WSDL and templates in this paper.

Based on the WSDL extensions and templates, development tools can easily be created to generate SSOT service source code with corresponding client access code. These tools can simplify development for programmers, which encourages the use of SSOT services to replace data-replication. Eclipse is an excellent platform to implement such tools. There are three additional future work topics.

Maintenance and Upgrades: Every active service goes through changes. Besides regular change management, services may experience unexpected emergency incidents. When these incidents occur, the clients should be alerted about their occurrences, side-effects, recovery progress and estimated recovery times. As suggested by Ludrig *et al.* [162] a subscription service can be used to communicate change, maintenance and emergency notices. Protocols and language extensions can be defined for various types of change, maintenance and emergency activities. Since SSOT introduces a single point of failure, a cloud infrastructure specifically designed for SSOT can improve its availability.

Local Temporary Cache: If the cost of the SSOT service invocation is a concern, clients can cache the SSOT resources temporarily. Once a SSOT resource is obtained, there is a good chance that the client needs the same SSOT resource for related processes. Therefore, temporarily caching the SSOT resource will likely reduce service invocations. The cached SSOT resource can be flushed after a timeout period. The SSOT update or deletion message should also flush the related resource from the cache. The caching functionality would ideally be implemented as middleware that supports the SSOT service architecture.

Schema Synchronization: For existing applications to adopt a SSOT service model, the existing client applications need to map their local data to the SSOT data. Both the SSOT and the clients can make use of existing ontology studies, which define data syntax and semantics for specific industries. For example, HL7 [163] is defined for the health industry; RosettaNet [164] is defined for e-business; EDIFACT [165] is defined for electronic data interchange. Moving toward standard languages will benefit the survival and the long term growth of the industry. The SSOT service model does not define the communication architecture or protocol between the clients (e.g. between the clinic application and the pharmacy application in the EPR use case), but adopting the SSOT service model can simplify communication between the clients. For example, the pharmacy can verify a patient's prescription with the clinic by the patient SSOT-ID and avoid multiple drug dispensing.

9.7 Conclusion

The SSOT service model described in this paper addresses the data-synchronization problems that arise due to data-layer replication across distributed systems. On the other hand, the SSOT service model introduces a single point of failure in the system. Depending on the Service Level Agreement (SLA), the SSOT may need support from multi-site configurations or cloud-infrastructure with fail-over capability. Although the data-replication model does not have a single point of failure, it suffers from data-synchronization and data-inconsistency issues. Data synchronization usually involves defining a custom peer-to-peer data exchange agreement. The custom agreement tightly couples the data provider and consumer, which makes switching providers very costly. Nonetheless, data synchronization usually happens during the overnight maintenance windows. Data becomes stale between synchronizations. Furthermore, data replication keeps a full copy of the provider's data at the clients' sites. If clients use only a small portion of the provider's data, then the clients are wasting resources. An added benefit of the SSOT service model is that it can control what data each client is authorized to access, while data replication makes all data available to clients.

The SSOT service model allows the provider and clients to be loosely coupled. Clients do not need to pledge infrastructure resources for the foreign data. The SSOT service model also provides up-to-date data. Overall, we believe that the SSOT service model can be used to eliminate data replication, enforce data autonomy, advocate data self-containment, ease data maintenance and enhance data protection. In the long term, these properties will also increase business adaptability.

Within large enterprises or government agencies, managing large amounts of data as a single entity is problematic. Decomposing a large data set into smaller autonomous and independently managed data sets can increase flexibility. As in the EPR case, once the EPR SSOT service is established, a new patient related service can be created without defining and creating its own patient data set. The new service does not need to negotiate with other parties regarding data acquisition or synchronization. The new service can loosely couple with the EPR SSOT and be established quickly. In addition, the SSOT service model allows each individual service to be self-contained and maintain its local database. For example, the clinic application and the pharmacy application in the EPR use case maintain their individual local databases without sharing data with the EPR system. This characteristic is very important in the health industry, where patients' privacy is closely monitored.

The SSOT service model is also applicable to the financial industry. Banking, investment and insurance businesses are often integrated under one corporation. However, legislation may require each of these businesses to be separate entities. The SSOT service model allows the corporation to create a customer SSOT to register each customer once. Banking, investment and insurance services can run as separate entities, while being loosely coupled with the customer SSOT service. With the SSOT service model, new financial services can be introduced more quickly. Similarly, the SSOT service model can benefit any jurisdiction that provides multiple services.

10 What Do Programmers Know about Software Energy Consumption?

As described in Section 2.2.1, the 2015 DZone Guide to Continuous Delivery [56] reported 50% of the IT practitioners had employed DevOps in their organizations, and the adoption rate kept rising. In 2017, the State of DevOps [11] reported 24% of the organizations employed DevOps ran their production systems in the public cloud; 18% would move to the public cloud within six months; 13% would move within one year. Therefore, the energy consumption of the data centers, which provide public cloud service, continually increases. I wondered whether programmers are aware of the energy consumed by their software running on the cloud. Therefore, I surveyed programmers accordingly, and published the results in the article "What Do Programmers Know about Software Energy Consumption?" [38].

The article "What Do Programmers Know about Software Energy Consumption?" [38] was published in the IEEE Software Magazine, May-June 2016. The paper is reprinted in this chapter, with permission, from Candy Pang, Abram Hindle, Bram Adams, and Ahmed E. Hassan @2016 IEEE.

The major contribution of this publication is studied these following research questions:

- Are programmers aware of software energy consumption?
- What do they know about reducing energy consumption?
- What's their level of knowledge about energy consumption?
- What do they think that causes high energy consumption?

By surveying 122 programmers, I concluded the following findings:

1. Programmers have limited awareness of software energy consumption
2. Programmers lack knowledge about how to reducing software energy consumption
3. Programmers lack knowledge of software energy consumption
4. Programmers are unaware of software energy consumption's causes

With the research findings, many works can be done to monitor and reduce software energy consumption, but most importantly programmers should be made aware about software energy consumption and how to reduce consumption.

For this publication, I was responsible for the design of the research program, the construction of surveys, the collection of data, the analysis of the research data, and manuscript composition. Dr. Abram Hindle, Dr. Bram Adams and Dr. Ahmed E. Hassan were involved with analysis of the research data, concept formation and manuscript composition.

Impact: As of writing, What Do Programmers Know about Software Energy Consumption has been cited 58 times according to the Google Scholar [85]. The citations include different contexts, such as empirical conference paper that studied software engineers' perspective on software energy consumption [166], and journal article that studied energy consumption in computer clusters, grids, and clouds [167].

Abstract

A survey revealed that programmers had limited knowledge of energy efficiency, lacked knowledge of the best practices to reduce software energy consumption, and were unsure about how software consumes energy. These results highlight the need for training on energy consumption.

10.1 Introduction

With the rising popularity of mobile computing and the advent of large-scale cloud deployments, the nonfunctional requirement of minimizing software energy consumption has become a concern. For mobile devices, energy consumption affects battery life and limits device use. For datacenters, energy consumption limits the number of machines that can be run and cooled. According to an IDC white paper, "Today, for every $1.00 spent on new hardware, an additional $0.50 is spent on power and cooling, more than double the amount of five years ago. Datacenters at their power and cooling thresholds are unable to support new server deployments, a fact that severely limits the expansion of IT resources [168]."

Unfortunately, the demand for energy-efficient computing isn't reflected in the education, training, or knowledge of programmers. Programmer training often focuses on methodologies such as object-oriented programming and nonfunctional requirements such as performance. Performance optimization is often considered a substitute for energy optimization because a faster system likely consumes less energy. Although this is a step in the right direction, it's insufficient and sometimes even incorrect. For instance, parallel processing might improve performance by reducing calculation time. However, saving and restoring execution context, scheduling threads, and losing locality might end up consuming more resources than sequential processing [169].

A previous analysis based on energy-related questions on StackOverflow (http://stackoverflow.com) showed that programmers had many such questions but rarely got appropriate advice [170]. To gain more tangible evidence of and concrete insight into this problem, we surveyed programmers to gauge their knowledge of software energy consumption and efficiency. In particular, we addressed four questions. Are programmers aware of software energy consumption? What do they know about reducing it? What's their level of knowledge about it? What do they think causes spikes in it?

10.2 The Survey

An anonymous online survey comprised 13 questions in four phases (full survey details, data, and analysis can be found online [171]).

Phase 1 involved three questions regarding respondent demographics:

- How many years of programming experience do you have?
- How would you rank your programming skill--beginner, intermediate, or advanced?
- In what programming language are you most proficient?

Phase 2 involved eight quantitative questions. The first two evaluated the respondents' knowledge of software energy consumption (using the common term "power consumption" [172]):

- For desktop computers, rank the software power consumption of the CPU, hard drive, memory, network, and screen and GPU.
- For mobile devices, rank the software power consumption of the CPU, data storage device, memory, network, and screen and GPU.

The next six yes/no questions gathered information about the respondents' experience with software energy consumption:

- Do you take power consumption into account when developing software?
- Is minimizing power consumption a requirement or a concern of your software?
- Have users complained about your software's power consumption?
- Have you modified your software to reduce power consumption?

- Have you measured your software's power consumption? If yes, how do you measure it? (If the respondent answered yes, the questionnaire provided additional space for a text response.)
- Would power consumption be one of your decision factors when choosing a mobile development platform?

Phase 3 involved two qualitative questions, allowing respondents to further express their knowledge and experience regarding software's power consumption:

- What software functions have higher power consumption?
- How would you improve your software's power efficiency?

Phase 4 involved optional qualitative follow-up interviews.

We posted survey invitations in numerous programming-related Reddit (www.reddit.com) subgroups (subreddits) between 20 August and 4 September 2013. We received 122 responses and conducted four follow-up interviews.

10.3 **The Results**

The survey respondents identified themselves as programmers. Of the 122 respondents, 37 (30 percent) used C or C++ and 84 (69 percent) used C#, Java, JavaScript, Perl, PHP, Python, or Ruby. According to the November 2014 TIOBE Index (www.tiobe.com/index.php/tiobe_index), these languages accounted for 54 percent of existing programs and likely represented more than half the software running in datacenters. In addition, Java is the primary language for Android and BlackBerry, whereas C# is the primary language for Windows Phones.

10.3.1 Programmers Have Limited Awareness of Software Energy Consumption

Our survey results show that the programmers rarely addressed energy efficiency and that users rarely requested it. Only 22 respondents (18 percent) claimed to take energy consumption into account when developing software. Only 17 respondents (14 percent) considered minimizing energy consumption a requirement. Twenty-six respondents (21 percent) said they modified software to reduce energy consumption.

One interviewee indicated that clients "care first and foremost about speed of development, and secondly about reasonable quality and performance." This suggests that the lack of attention to software energy consumption is an issue of priorities.

These results show that these programmers either were unaware of energy efficiency or weren't asked to address it. An interviewee mentioned that "1 watt would be a lot of power for a mobile phone, [but] it's absolutely negligible in comparison to other household appliances." That 1 watt might be negligible on the personal level, but on the global level, energy consumed by all mobile devices and datacenters multiplies. In 2006, 6,000 US datacenters reportedly consumed 61 billion kilowatt-hours of energy costing US$4.5 billion [173].

Similarly, software users and clients were unaware of software energy consumption. Only 4 respondents (3 percent) reported that their users complained about their software's energy consumption.

Our results confirmed Hammad Khalid and his colleagues' finding that mobile-application users have low awareness of resource use [174]. Their results showed that resource related complaints (application reviews), including energy consumption complaints, ranked last out of 12 types of user complaints in terms of frequency. Although users don't complain frequently about resource consumption, Khalid and his colleagues' results also show that resource-related complaints negatively affect users. So, despite the low frequency of such complaints, they're highly troubling. If customers and clients aren't asking for energy-efficient software, programmers are less likely to address the energy efficiency of software. Hence, appropriate public education and expanded awareness among clients and programmers about software energy consumption is needed.

10.3.2 Programmers Lack Knowledge of Reducing Software Energy Consumption

To reduce software energy consumption, programmers must start by measuring the energy consumption of their software. Only 12 respondents (10 percent) said they did this. Fifteen respondents (12 percent) indicated that you can measure software energy consumption through a power meter, the battery, the power supply, resource measurement, software tools, and CPU time.

These results show that these programmers lacked knowledge of how to accurately measure software energy consumption. Most of the suggested methods measure the overall hardware energy consumption, not the fine-grained energy consumption of the software. In addition, mobile device batteries don't accurately report the actual energy use [172]. Ding Li and his colleagues also found that programmers used "typical practices in energy measurement studies ..., [which] have limitations that could introduce inaccuracy [175]."

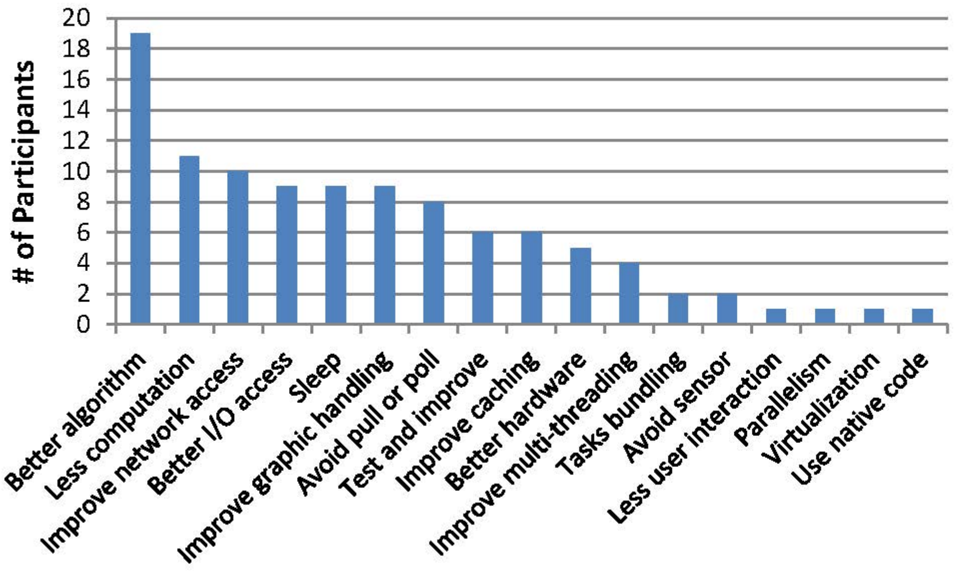


Figure 10.1 Respondents' responses regarding ways to improve software energy efficiency. The respondents' answers varied widely, indicating the need for increased education on software energy consumption and efficiency.

Measuring software energy consumption is a challenge. One interviewee stated that "one has to have a proper understanding of the entire system [to] make an informed [energy consumption] analysis." Programmers have to understand the interactions between high-level and low-level components to really analyze the root causes of software energy consumption. The survey and interviews showed that most of the respondents had difficulty measuring and optimizing software energy consumption even when the appropriate tools were available.

Another interviewee admitted, "It's more often the hardware rather than the software that we are interested in when we talk about energy consumption." Although few respondents measured the energy consumption of their software, 79 (65 percent) of them considered energy consumption as a factor when choosing a mobile development platform. Many respondents relied on choosing the right platform and hardware to ensure the energy efficiency of their software. However, they rarely addressed software energy consumption.

Figure 10.1 summarizes how the respondents would improve energy consumption. Nineteen respondents (16 percent) were aware that better algorithms lead to better energy efficiency, which was the most popular suggestion. Only 11 (9 percent) and 8 (7 percent) of the respondents, respectively, were aware that less computation and reduced polling can reduce energy consumption. The results show that the respondents' ideas about how to best reduce software energy consumption varied widely. Furthermore, university courses don’t often teach about the link between better algorithms and energy consumption.

10.3.3 Programmers Lack Knowledge of Software Energy Consumption

Figure 10.2 summarizes the respondents’ rankings of the software energy consumption for desktop computer and mobile-device components.

For desktop computer components, we expected these rankings (from highest to lowest consumption):

1. CPU,
2. hard drive,
3. screen and GPU,
4. network, and
5. memory.

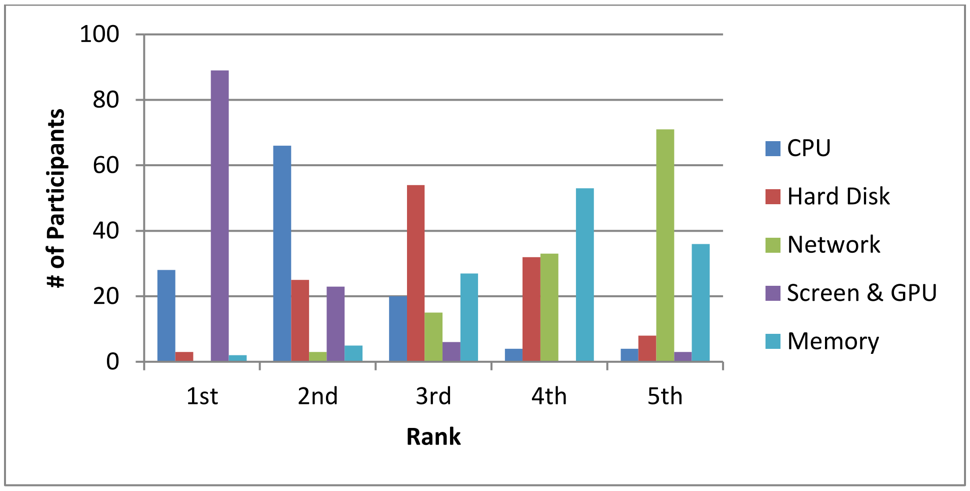
For mobile-device components, we expected these rankings:

1. screen and GPU,
2. CPU,
3. network,
4. hard drive, and
5. memory.

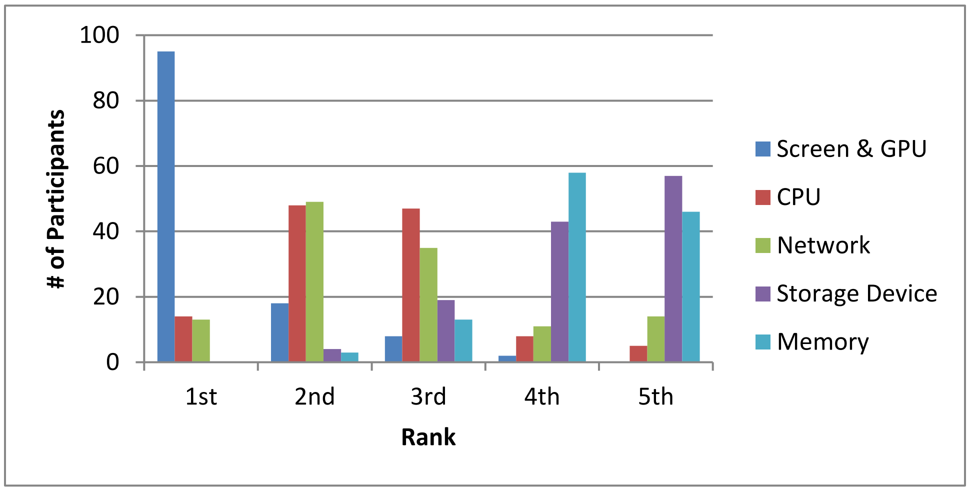
We based these rankings on current conventional wisdom, backed up by experiments we had performed over the past three years and recent studies [172] [175] [176]. Because these rankings weren’t written in stone, they could differ among specific hardware. We were focusing on the consistency of rankings across all respondents, independent of our expected ranking. For desktop computers, only 1 respondent (1 percent) ranked the components in our expected order. For mobile devices, 12 respondents (10 percent) ranked the components in our expected order.

Using Spearman’s rank correlation, we compared the respondents’ rankings with the expected ranking. Generally, positive correlations closer to 1 indicate stronger agreement. If two rankings completely match, their correlation is 1. If they’re the inverse of each other, the correlation is -1. If they’re unrelated, a correlation near 0 is possible.

For desktop computers, the average correlation between the respondents’ rankings and the expected ranking was 0.48, indicating a medium level of agreement. For mobile devices, the



(a) Desktop Computer Components Ranked by Energy Consumption



(b) Mobile Device Components Ranked by Energy Consumption

Figure 10.2 Respondents' rankings of energy consumption. Considerable disagreement existed on whether a particular component consumed more energy than another.

average correlation was 0.75, indicating a much stronger agreement. The correlation's standard deviation was 0.25 for desktop computers and 0.20 for mobile devices.

We also used Spearman's rank correlation to compare the respondents' rankings against each other (interagreement), regardless of the expected ranking. The correlation was 0.3 for desktop computers and 0.6 for mobile devices. So, respondents had less internal agreement on the energy consumption of desktop computer components than on the consumption of mobile device components. The correlation's standard deviation was 0.48 for desktop computers and 0.32 for mobile devices. This implies that respondents agreed less about the energy consumption of desktop hardware components and more about the energy consumption of mobile-device components.

In other words, considerable disagreement existed on whether a particular component consumed more energy than another. One explanation might be that different types of programmers make different assumptions about the energy consumption of hardware components. For example, game programmers interact mostly with the screen and GPU, so they're more likely to identify the screen and GPU as the most energy consuming components. Programmers might blame the most obvious component without understanding how software consumes energy.

Furthermore, programmers might focus overly on their users' on-screen experience--that is, on what's observable. The respondents overwhelmingly ranked the screen and GPU as the highest-energy-consuming components: 82 respondents (67 percent) for desktop computers and 95 respondents (78 percent) for mobile devices. It is true, though, that the screen and GPU often consume the most energy on mobile devices.

The overall results show that programmers lack consistent knowledge regarding the energy consumption relationship between software and hardware. Nonetheless, programmers have more consistent knowledge about software energy consumption on mobile devices than on desktop computers. So, it might be more effective to develop education and awareness programs and guidelines for specific domains (for example, mobile devices and gaming).

10.3.4 Programmers Are Unaware of Software Energy Consumption's Causes

Gustavo Pinto and his colleagues mined Stack Overflow data to identify seven causes of unnecessary software energy consumption [170]:

- unnecessary resource use,
- faulty GPS behavior,
- background activities,
- excessive synchronization,
- background wallpapers,
- advertisements, and
- high GPU use.

Our results closely matched the results of Pinto and his colleagues. Taken as a group, the respondents identified most of Pinto's seven causes. For example, in Figure 10.3, "Network" refers to unnecessary resource use, "Sensor" refers to faulty GPS behavior, "Threading" refers to

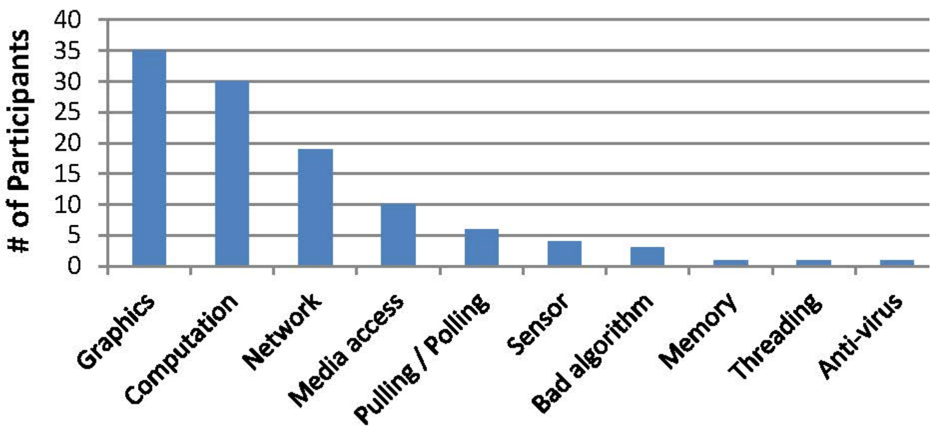


Figure 10.3 Respondents' responses regarding causes associated with high energy consumption. Only a few of the respondents could identify those causes.

background activities, "Pulling or polling" refers to excessive synchronization, and "Graphics" refers to background wallpapers and high GPU use. The respondents didn't identify advertisements.

However, individually, only 19 respondents (16 percent) identified network data access as a cause of high energy consumption. The low identification rate matches the observation of Li and his colleagues [175]. Six respondents (5 percent) identified pulling or polling for excessive synchronization. Only 4 (3 percent) identified sensor use, which can leave the GPS turned on for too long. Thirty-five (29 percent) identified graphics as a cause of high energy consumption for functions such as background wallpapers and animations. In short, the numbers show that only a few of the respondents could identify the causes of high energy consumption.

10.4 Limitations

One limitation was our use of social media to recruit survey respondents, instead of directed emails or phone calls. Nevertheless, through social media, we were able to reach programmers in the field whom we otherwise couldn't have reached. More than half a million users have subscribed to programming-related subreddits, which often have more than 1,000 concurrent users. To avoid having our survey invitation tagged as spam and to avoid negative reactions, we posted it to one relevant subreddit at a time. This process significantly lengthened the data-gathering period and limited the number of respondents.

Another limitation is that we didn't control for the development process that the respondents used. Many development processes exist. An enterprise might employ the full

COBIT (Control Objectives for Information and Related Technology; www.isaca.org/cobit) development cycle. A mid-size shop might use agile processes. A one-man startup might do whatever is necessary. In a formal process, programmers might not have the opportunity to specify the functional or nonfunctional requirements. But surveying a statistically significant number of programmers for each process type would have been difficult. So, we surveyed a board range of programmers. Future studies need to investigate the energy consumption knowledge of IT workers in different roles.

A third limitation is that we focused on programmers' software energy consumption knowledge. Software has many nonfunctional requirements (for example, memory use, performance, security, and usability); energy consumption is just one of them, albeit the least-studied one. However, although nonfunctional requirements might affect each other, mixing other criteria into our study might have blurred the results.

10.5 **Conclusion**

The programmers in our study lacked knowledge and awareness of software energy-related issues. More than 80 percent of them didn't take energy consumption into account when developing software. Nevertheless, most of them considered software energy consumption to be important when choosing a mobile development platform.

The fact that only 3 percent of the respondents received complaints about software energy consumption might suggest that users are unaware of it. As Chenlei Zhang and his colleagues argued, the creation of benchmarks and reporting mechanisms (similar to Energy Star) that inform users of software energy efficiency can significantly increase user awareness [177]. Increased user awareness will, in turn, motivate programmers to measurably enhance their software's energy efficiency. As one Reddit respondent commented, the "survey has at least made me consider … possible costs of doing things."

Pinto and his colleagues identified eight strategies to reduce energy consumption through software modification [170]:

- minimizing IO,
- bulk operations,
- avoiding polling,
- hardware coordination,
- concurrent programming,
- lazy initialization,
- race to idle, and
- efficient data structure.

These strategies should be part of programmers' education. In addition, development tools can be created to identify unnecessary energy consumption and suggest how to reduce it. Educators could develop slides, videos, projects, and assignments as part of an undergraduate curriculum for energy efficiency and sustainability.

11 Final Words

In this dissertation, I have presented four of my research results in the areas of DevOps education, maintaining continuity, single source of truth for microservices, and software energy consumption. These topics may seem disconnected, but they all linked to the popular DevOps movement in the IT industry. The diversity reflects that DevOps is comprehensive and covers a broad range of IT aspects. Therefore, there are many DevOps research opportunities.

11.1 **Lessons Learned**

First, my research (Chapter 1 to 7) shows that academic institutions have not embraced DevOps education. Since DevOps is becoming an industrial standard, institutions should incorporate DevOps knowledge and skills in undergraduate computer science curriculum to prepare students for industry. However, DevOps is not mature enough for systematic teaching. Therefore, institutions can only teach basic DevOps knowledge to get students ready for advancing their DevOps skills in industry. I proposed 11 hypotheses to serve as a base for academic DevOps education.

Second, my research (Chapter 8) shows that there are many continuous processes in DevOps. The continuous processes need proper maintenance to sustain continuity. Therefore, I proposed continuous maintenance to maintain repositories and artifacts properly and consistently through automation, summarization, compaction, archival, and removal during pre- and post-production. Continuous maintenance is necessary to make DevOps sustainable.

Third, my research (Chapter 9) focuses on interactions between microservices. Since "microservices and DevOps go hand-in-hand" [35], microservices have become popular along DevOps. DevOps allows companies to "respond faster to business needs" [3]. For microservices to be swift along with DevOps, they have to be single sources of truth, that are deployable individually. Then, microservices interact with each other to achieve business purposes. I defined the mutable and non-mutable interactions between microservices that are the basis of all microservices interactions. With microservices, DevOps can attain its advantage of being responsive.

Fourth, my research (Chapter 10) shows that programmers lack knowledge about software energy consumption for effective datacenter deployment. The majority of organizations that employ DevOps will run their production systems in public datacenters [11]. However, the limited power and cooling thresholds of the datacenters severely limits the expansion of IT resources [168]. Concerning the limited power supply in datacenter, I surveyed programmers about their knowledge of software energy consumption. My survey shows that programmers lack knowledge and awareness of software energy-related issues. More than 80 percent of the surveyed programmers do not consider energy consumption when developing software. Programmers need to be more aware about software energy consumption to keep datacenters sustainable for future DevOps expansion.

11.2 Future Work

In addition to the future work mentioned so far, there are many more DevOps research topics worth exploring, especially DevSecOps [47] [48]. Lately, DevSecOps has become a focus of the DevOps community, in blogs [178], webinars [179], surveys [49] [50], conferences [23], writings [180] [181], and more. Many DevOps practitioners have "felt the struggle that comes from trying to balance speedy DevOps practices with necessary security practices" [180]. Many industrial researchers are studying how to incorporate security into DevOps. Academic institutions should also study how to incorporate DevSecOps into their security courses. "DevSecOps is really becoming mandatory. We’ve kind of known this was coming for years, but it’s really starting to come home for everyone" [181]. Studying DevSecOps provides an opportunity for academic researchers to be leaders in an emergent and highly relevant area.

In conclusion, DevOps is influential in industry and provides many research opportunities for academia. Academics should put more effort into DevOps research.

Bibliography

[1] XebiaLabs, Inc., "2017 The Year DevOps Got Real," [Online]. Available: https://xebialabs.com/assets/files/whitepapers/DevOpsGetsReal.pdf. [Accessed 16 03 2018].

[2] D. A. Tamburri and D. Perez-Palacin, "Special Issue: DevOps Quality Engineering," *Journal of Software: Evolution and Process,* pp. 1-3, 2018.

[3] I. Sacolick, "What is devops? Transforming software development," DZone, Inc., 26 02 2018. [Online]. Available: http://www.infoworld.com/article/3215275/devops/what-is-devops-transforming-software-development.html. [Accessed 27 02 2018].

[4] W. Jackson, "2017 State of the Software Supply Chain," 2017. [Online]. Available: www.sonatype.com. [Accessed 20 04 2018].

[5] M. Werner, "The 2018 DZone Guide to DevOps Culture and Process Volume V," DZone, Inc., 2018. [Online]. Available: https://dzone.com/storage/assets/7828456-dzone2018-researchguide-devops.pdf. [Accessed 16 01 2018].

[6] M. Schmidt, "DevOps and Continuous Delivery: Not the Same," DevOps.com, MediaOps, LLC., 08 04 2016. [Online]. Available: https://devops.com/devops-and-continuous-delivery-not-same/. [Accessed 12 04 2016].

[7] A. Streets, "Dedicated DevOps? I Smell a Rat…," DZone, Inc., 25 07 2017. [Online]. Available: https://dzone.com/articles/dedicated-devops-i-smell-a-rat. [Accessed 28 07 2017].

[8] J. Lee, "Can DevOps Be a Role?," DZone, Inc., 24 08 2017. [Online]. Available: https://dzone.com/articles/can-devops-be-a-role. [Accessed 30 08 2017].

[9] xMatters & Atlassian, "DevOps Maturity Survey Report 2017," 2017. [Online]. Available: http://info.xmatters.com/rs/178-CPU-592/images/atlassian_devops_survey.pdf. [Accessed 02 09 2017].

[10] A. Wallgre, "You Can't Buy DevOps in a Box," DZone, Inc., 26 08 2017. [Online]. Available: https://dzone.com/articles/you-cant-buy-devops-in-a-box-electric-cloud. [Accessed 26 08 2017].

[11] E. Bruno, "2017 State of DevOps Research Reports," Interop ITX, UBM LLC., 2018. [Online]. Available: http://images.reg.techweb.com/Web/UBMTechweb/%7Bcc7768e4-e40f-4033-b4b0-f55f2bb5222a%7D_Interop_ITX_State_of_DevOps_FinalFinal.pdf. [Accessed 25 01 2018].

[12] DevOps Institute, "DevOps Institute," [Online]. Available: https://devopsinstitute.com.

[13] E. Oehrlich and J. Groll, "2019 Upskilling: Enterprise DevOps Skills Report," 03 2019. [Online]. Available: https://devopsinstitute.com/wp-content/uploads/2019/03/UpskillingReport-Final.pdf. [Accessed 06 03 2019].

[14] Puppet & IT Revolution, "2015 State of DevOps Report," 2015. [Online]. Available: https://puppet.com/system/files/2016-03/2015-state-of-devops-report.pdf. [Accessed 17 06 2016].

[15] A. Brown, N. Forsgren, J. Humble, N. Kersten and G. Kim, "2016 State of DevOps Report," Puppet & DORA, 2016. [Online]. Available: https://puppet.com/resources/white-paper/2016-state-devops-report/. [Accessed 10 09 2016].

[16] N. Forsgren, J. Humble, G. Kim, B. Alanna and N. Kersten, "2017 State of DevOps Report," Puppet & DORA, 2017. [Online]. Available: https://puppet.com/system/files/2017-06/2017-state-of-devops-report_3.pdf. [Accessed 15 05 2017].

[17] ICAgile, "ICAgile Certified Professional in Foundations of DevOps (ICP-FDO)," [Online]. Available: https://icagile.com/Learning-Roadmap/DevOps/Foundations-of-DevOps. [Accessed 18 12 2017].

[18] ICAgile, "ICAgile Certified Professional Implementing DevOps (ICP-IDO)," [Online]. Available: https://icagile.com/Learning-Roadmap/DevOps/Implementing-DevOps. [Accessed 18 12 2017].

[19] ICAgile, "ICAgile Certified Expert In DevOps (ICE-DO)," [Online]. Available: https://icagile.com/Learning-Roadmap/DevOps/Expert-in-DevOps. [Accessed 19 12 2017].

[20] ASPE Training, "ASPE Training," [Online]. Available: http://aspetraining.com/.

[21] Coursera Inc., "Coursera," [Online]. Available: https://www.coursera.org/.

[22] Skillsoft Corporation, "Skillsoft," [Online]. Available: https://www.skillsoft.com/.

[23] Sonatype Inc., "All Day DevOps," [Online]. Available: https://www.alldaydevops.com/.

[24] CloudBees Inc., "DevOps World - Jenkins World," [Online]. Available: https://www.cloudbees.com/devops-world.

[25] IT Revolution, "DevOps Enterprise Summit (DOES)," [Online]. Available: https://events.itrevolution.com/.

[26] DevOps.com, MediaOps, LLC., "DevOps Experience Virtual Summit," [Online]. Available: https://devopsexperience.io/.

[27] IBM Canada Laboratory, "Centers for Advanced Studies Conference (CASCON)," [Online]. Available: https://www-

01.ibm.com/ibm/cas/cascon/index.jsp.

[28] ACM SigSoft, "International Workshop on Continuous Software Evolution and Delivery (CSED)," International Conference on Software Engineering (ICSE), 14 05 2016. [Online]. Available: http://continuous-se.org/CSED2016/.

[29] E. Novoseltseva, "Top 16 DevOps Blogs You Should Be Reading," DZone, Inc., 18 02 2018. [Online]. Available: https://dzone.com/articles/top-16-devops-blogs-you-should-be-reading. [Accessed 21 02 2018].

[30] DZone, Inc., "DZone Integration Zone - Enterprise Integration news, tutorials, tools," [Online]. Available: https://dzone.com/enterprise-integration-training-tools-news.

[31] DZone, Inc., "DZone DevOps Zone - DevOps news, tutorials, tools & reviews," [Online]. Available: https://dzone.com/devops-tutorials-tools-news.

[32] DevOps.com, MediaOps, LLC., "Where The World Meets DevOps," [Online]. Available: https://devops.com/.

[33] DevOps Research and Assessment LLC., "DevOps Research and Assessment (DORA)," [Online]. Available: https://devops-research.com/.

[34] C. Pang and A. Hindle, "Continuous Maintenance," in *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Raleigh, NC, USA, 2016.

[35] P. Belagatti, "7 Microservices Architecture Books Every DevOps Enthusiast Must Read," DZone, Inc., 27 04 2018. [Online]. Available: https://dzone.com/articles/7-microservices-architecture-books-every-devops-en. [Accessed 30 04 2018].

[36] M. Richards, Microservices vs. Service-Oriented Architecture, O'Reilly, 2016.

[37] C. Pang and D. Szafron, "Single Source of Truth (SSOT) for Service Oriented Architecture (SOA)," in *International Conference on Service-Oriented Computing*, Paris France, 2014.

[38] C. Pang, A. Hindle, B. Adams and A. E. Hassan, "What Do Programmers Know about Software Energy Consumption?," *IEEE Software,* vol. 33, no. 3, pp. 83-89, May-June 2016.

[39] P. M. Duvall, S. Matas and A. Glover, Continuous Integration: Improving Software Quality and Reducing Risk, Addison-Wesley, 2007.

[40] K. Beck, Extreme Programming Explained: Embrace Change, Addison-Wesley, 1999.

[41] M. A. Cusumano and R. W. Selby, Microsoft Secrets: How the world's most powerful software company creates technology, shapes markets, and manages people, New York: Simon and Schuster, 1998.

[42] M. A. Cusumano and R. W. Selby, "How Microsoft Builds Software," *Communications of the ACM,* vol. 40, no. 6, pp. 53-61, June 1997.

[43] J. Humbel and D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, Addison-Wesley, 2010.

[44] Kharnagy, "File:Devops-toolchain.svg," Wikipedia, 07 09 2016. [Online]. Available: https://en.wikipedia.org/wiki/File:Devops-toolchain.svg. [Accessed 11 03 2018].

[45] A. Still and P. Stanhope, DevOps and DNS - Improving Web Application Performance at the DNS Layer, O'Reilly, 2017.

[46] B. Fitzgerald and K.-J. Stol, "Continuous software engineering: A roadmap and agenda," *Journal of Systems and Software,* vol. 123, pp. 176-189, January 2017.

[47] DevSecOps, "DevSecOps - Security as Code," [Online]. Available: http://www.devsecops.org/.

[48] K. Carter, "Francois Raynaud on DevSecOps," *Computing Edge,* pp. 15-18, 01 2018.

[49] Sonatype Inc., "DevSecOps Community Survey 2017," 2017. [Online]. Available: https://www.sonatype.com/2017survey. [Accessed 25 03 2017].

[50] D. E. Week, "DevSecOps Community Survey," Sonatype Inc., 2018. [Online]. Available: https://www.sonatype.com/hubfs/SON_Survey2018_final.pdf. [Accessed 16 04 2018].

[51] J. Hand, ChatOps - Managing Operations in Group Chat, O'Reilly, 2016.

[52] A. Richardson, "GitOps: High-Velocity CI/CD for Kubernetes," 14 02 2018. [Online]. Available: https://dzone.com/articles/gitops-high-velocity-cicd-for-kubernetes. [Accessed 15 02 2018].

[53] V. Farcic, The DevOps 2.0 Toolkit - Automating the Continuous Deployment Pipeline with Containerized Microservices, CloudBees, 2016.

[54] C. Tozzi, "DevOps Is Killing the ITOps Engineer (and That's Bad)," DevOps.com, 12 07 2019. [Online]. Available: https://devops.com/devops-is-killing-the-itops-engineer-and-thats-bad. [Accessed 15 07 2019].

[55] A. Noller, "DZone's 2014 Guide to Continuous Delivery," DZone, Inc., 14 04 2014. [Online]. Available: https://dzone.com/articles/introducing-dzones-2014-guide. [Accessed 24 02 2018].

[56] J. Gopal, "The DZone Guide to Continuous Delivery," DZone, Inc., 2015. [Online]. Available: https://dzone.com/storage/assets/18140-dzone_2015continuousdelivery_9.pdf. [Accessed 29 03 2016].

[57] J. Esposito, "The DZone Guide to Continuous Delvery Volume III," DZone, Inc., 2016. [Online]. Available: https://dzone.com/storage/assets/1216574-dzone-continuousdelivery2016.pdf. [Accessed 29 03 2016].

[58] M. Werner, "The DZone Guide to DevOps, Continuous Delivery and Automation Volume IV," DZone, Inc., 2017. [Online]. Available: https://dzone.com/storage/assets/4152382-dzone-guidetodevopscontinuousdeliveryautomation.pdf. [Accessed 04 02 2017].

[59] N. Forsgren and M. Kersten, "DevOps Metrics," *ACM Queue,* vol. 16, no. 6, pp. 30:19-30:34, 11-12 2017.

[60] Puppet Inc., "Puppet," [Online]. Available: http://puppet.com.

[61] xMatters, "(x)matters," [Online]. Available: http://www.xmatters.com.

[62] Interop ITX, UBM LLC., "Interop ITX," [Online]. Available: https://www.interop.com.

[63] J. Groll, "DevOps Track," Interop ITX, UBM LLC., 30 04 2018. [Online]. Available: https://www.interop.com/devops. [Accessed 08 03 2018].

[64] Interop ITX, UBM LLC., "DevOps Foundation Certification Course," 30 04 2018. [Online]. Available: https://www.interop.com/devops-certification-course. [Accessed 08 03 2018].

[65] XebiaLabs, Inc., "XebiaLabs Enterprise DevOps," [Online]. Available: https://xebialabs.com/.

[66] R. Stroud, C. Gardner, E. Oehrlich, E. Klavens, A. Kinch and D. Lynch, "The Forrester Wave: Continuous Delivery And Release," 2017. [Online]. Available: https://xebialabs.com/assets/files/analyst-reports/The_Forrester_Wave_Continuous_Delivery_and_Release_Automation_2017.pdf. [Accessed 01 03 2018].

[67] C. Fletcher and L. F. Wurster, "Magic Quadrant for Application Release Automation," Gartner Inc., 2017.

[68] XebiaLabs, Inc., "Periodic Table of DevOps Tools (v2)," 14 06 2016. [Online]. Available: https://blog.xebialabs.com/2016/06/14/periodic-table-devops-tools-v-2/. [Accessed 24 02 2018].

[69] XebiaLabs, Inc., "XebiaLabs Unveils Periodic Table of DevOps Tools v.3 at DevOps Enterprise Summit London 2018," 25 06 2018. [Online]. Available: https://xebialabs.com/company/press/xebialabs-unveils-periodic-table-of-devops-tools-v3-at-devops-enterprise-summit-london-2018/. [Accessed 26 04 2019].

[70] XebiaLabs, Inc., "Periodic Table of DevOps Tools (v3)," 25 06 2018. [Online]. Available: https://xebialabs.com/assets/files/infographics/periodic-table-of-devops-tools-v3.pdf. [Accessed 26 04 2019].

[71] XebiaLabs, Inc., "The Ultimate DevOps Tool Chest," [Online]. Available: https://xebialabs.com/the-ultimate-devops-tool-chest/. [Accessed 12 04 2018].

[72] Linus Torvalds, "Git," [Online]. Available: https://git-scm.com/.

[73] GitHub Inc., "GitHub," [Online]. Available: https://github.com.

[74] GitLab Inc., "GitLab," [Online]. Available: https://about.gitlab.com/.

[75] J. Noehr, "Bitbucket," Atlassian, 2008. [Online]. Available: https://bitbucket.org.

[76] Travis CI, GmbH, "Travis CI," [Online]. Available: https://travis-ci.org/.

[77] YAML.org, "YAML Ain't Markup Language," [Online]. Available: http://yaml.org/start.html.

[78] CloudBees Inc., "Jenkins," [Online]. Available: https://jenkins.io/.

[79] Docker Inc., "Docker," [Online]. Available: https://www.docker.com/.

[80] The Linux Foundation, "Kubernetes," [Online]. Available: https://kubernetes.io/.

[81] J. La Torre, "50+ Useful Docker Tools," DZone, Inc., 01 01 2018. [Online]. Available: https://dzone.com/articles/50-useful-docker-tools. [Accessed 02 01 2018].

[82] S. Yegulalp, "What is Kubernetes? Container orchestration explained," InfoWorld, 04 04 2018. [Online]. Available: https://www.infoworld.com/article/3268073/containers/what-is-kubernetes-container-orchestration-explained.html. [Accessed 06 04 2018].

[83] IEEE, "IEEE Xplore Digital Library," [Online]. Available: http://ieeexplore.ieee.org.

[84] Association for Computing Machinery (ACM), "ACM Digital Library," [Online]. Available: http://dl.acm.org.

[85] Google Inc., "Google Scholar," [Online]. Available: http://scholar.google.com.

[86] F. Erich, C. Amrity and M. Danevaz, "A Qualitative Study of DevOps Usage in Practice," *Journal of Software: Evolution and Process,* 28 06 2017.

[87] B. B. N. de Franca, H. J. Jeronimo and G. H. Travassos, "Characterizing DevOps by Hearing Multiple Voices," in *Proceedings of the 30th Brazilian Symposium on Software Engineering (SBES)*, 2016.

[88] V. Garousi and M. Felderer, "Industrial and Academic Focus Areas in Software Testing," *IEEE Software,* vol. 34, no. 5, pp. 38-45, Sep/Oct 2017.

[89] A. Strauss and J. Corbin, Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory, Second ed., Sage Publication, 1998.

[90] B. G. Glaser and A. L. Strauss, The Discovery of Grounded Theory: Strategies for Qualitative Research, Aldine Transaction, 1967.

[91] K.-J. Stol, P. Ralph and B. Fitzgerald, "Grounded Theory in Software Engineering Research: A Critical Review and Guidelines," in *Proceedings of the 38th International Conference on Software Engineering (ICSE)*, Austin, Texas, 2016.

[92] K. Charmaz, Constructing Grounded Theory - A Practical Guide through Qualitative Analysis, SAGE Publications, 2006.

[93] S. Adolph, P. Kruchten and W. Hall, "Reconciling perspectives: A grounded theory of how people manage the process of software development," *The Journal of Systems and Software,* vol. 85, pp. 1269-1286, 8 February 2012.

[94] M. Hilton, N. Nelson, T. Tunnell, D. Marinov and D. Dig, "Trade-Offs in CI: Assurance, Security and Flexibility," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, Paderborn, Germany, 2017.

[95] M. Hilton, T. Tunnell, K. Huang, D. Marinov and D. Dig, "Usage, costs, and benefits of continuous integration in open-source projects," in *31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2016.

[96] W. P. Luz, G. Pinto and R. Bonifacio, "Building a Collaborative Culture: A Grounded Theory of Well Succeeded DevOps Adoption in Practice," in *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2018.

[97] D. Karanth and R. Williams, "Continuous Integration Part 3: Best Practices," DZone, Inc., 06 07 2017. [Online]. Available: https://dzone.com/articles/continuous-integration-part-3-best-practices. [Accessed 07 07 2017].

[98] M. Sahasrabudhe, "Why DevOps Efforts Fail," DZone, Inc., 22 08 2017. [Online]. Available: https://dzone.com/articles/why-devops-efforts-fail. [Accessed 24 08 2017].

[99] T. Randall, "Winning at Culture - The Keys to the Successful DevOps Organization," XebiaLabs, 10 05 2018. [Online]. Available: https://xebialabs.com/assets/files/whitepapers/WinningAtCulture_eBook_Final.pdf. [Accessed 23 05 2018].

[100] ACM & IEEE, "Computer Science Curricula 2013 - Curriculum Guidelines for Undergraduate Degree Programs in Computer Science," Association for Computing Machinery (ACM) and IEEE Computer Society, 2013.

[101] QS Top Universities, "2017 QS World University Rankings by Subject - Engineering and Technology," [Online]. Available: https://www.topuniversities.com/university-rankings/university-subject-rankings/2017/engineering-technology. [Accessed 03 08 2017].

[102] G. J. Myers, The Art of Software Testing, 1st ed., New York: Wiley, 1979, p. 177.

[103] The Australian National University, "COMP1040 The Craft of Computing," [Online]. Available: https://programsandcourses.anu.edu.au/2017/course/COMP1040. [Accessed 24 08 2017].

[104] University of Texas at Austin, "Curriculum - Department of Computer Science," [Online]. Available: https://www.cs.utexas.edu/undergraduate-program/academics/curriculum. [Accessed 17 08 2017].

[105] I. Kantor, "How to become a DevOps Engineering in six months or less," Medium.com, 03 06 2018. [Online]. Available: https://medium.com/@devfire/how-to-become-a-devops-engineer-in-six-months-or-less-366097df7737. [Accessed 09 07 2018].

[106] J. Lee, "Are You Qualified to Be a DevOps Engineer?," DZone, Inc., 12 09 2017. [Online]. Available: https://dzone.com/articles/are-you-qualified-to-be-adevops-enginee. [Accessed 15 09 2017].

[107] LinkedIn Corporation, "LinkedIn," [Online]. Available: https://www.linkedin.com/.

[108] C. Delanbanque, "7 Steps to DevOps Hiring Success," DevOps.com, MediaOps, LLC., 21 06 2017. [Online]. Available: https://devops.com/7-steps-devops-hiring-success/. [Accessed 09 01 2018].

[109] Wonster Worldwide, Inc., "Monster," [Online]. Available: https://www.monster.com/jobs.

[110] DZone, Inc., "DZone," [Online]. Available: https://dzone.com/pages/about.

[111] IEEE Computer Society, "IEEE Computer Society Jobs Board," [Online]. Available: https://computer.org/objs.

[112] J. Ambrosio, "DevOps Hiring: Look for the Right Mindset," InformationWeek, 03 10 2018. [Online]. Available: https://www.informationweek.com/devops/devops-hiring-look-for-the-right-mindset/d/d-id/1332949. [Accessed 09 10 2018].

[113] Project Management Institute, Inc., "Project Management Institute (PMI)," [Online]. Available: https://www.pmi.org/.

[114] International Institute of Business Analysis, "International Institute of Business Analysis (IIBA)," [Online]. Available: https://www.iiba.org/.

[115] ICAgile, "International Consortium for Agile (ICAgile)," [Online]. Available: https://icagile.com/.

[116] C. Garlick, D. DeGrandis, G. Gotimer and T. Guay, "ICAgile Learning Roadmap DevOps Track," ICAgile, 04 2018. [Online]. Available: https://icagile.com/Portals/0/LO%20PDFs/DevOps%20Learning%20Outcomes.pdf. [Accessed 27 07 2018].

[117] B. Simon, "Teaching Impacts of Technology in K-12 Education Specialization," University of California San Diego, [Online]. Available: https://www.coursera.org/courses?query=Teaching%20Impacts%20of%20Technology%20in%20K-12%20Education&. [Accessed 04 03 2019].

[118] ASPE Training, "DevOps Implementation Boot Camp (ICP-FDO)," [Online]. Available: http://aspetraining.com/courses/devops-implementation-boot-camp-icp-fdo. [Accessed 28 07 2018].

[119] Association for Computing Machinery (ACM), "International Conference on Software Engineering (ICSE)," [Online]. Available: http://www.icse-conferences.org/.

[120] The Edmonton Python User Group, "Edmonton.py," [Online]. Available: http://edmontonpy.com/.

[121] D. Edwards, "The (Short) History of DevOps," 17 09 2012. [Online]. Available: https://www.youtube.com/watch?v=o7-IuYS0iSE. [Accessed 13 04 2019].

[122] ACM Joint ESEC/FSE, "ESEC/FSE 2018 Artifacts," [Online]. Available: https://2018.fseconference.org/track/fse-2018-Artifacts. [Accessed 27 11 2018].

[123] A. Nigam and P. Nigam, "Citation Index and Impact factor," *Indian Journal of Dermatology, Venereology, and Leprology,* vol. 78, no. 4, pp. 511-16, July 2012.

[124] M. Branscombe, "AIOps: Is DevOps Ready for an Infusion of Artificial Intelligence?," The New Stack, 29 01 2019. [Online]. Available: https://thenewstack.io/aiops-is-devops-ready-for-an-infusion-of-artificial-intelligence/. [Accessed 04 02 2019].

[125] J. Ruohonen and V. Leppanen, "How PHP Releases Are Adopted in the Wild?," in *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*, 2017.

[126] M. K. Aljundi, Tools and Practices to Enhance DevOps Core Values, vol. M.Sc. Thesis, School of Business and Management, Lappeenranta University of Technology, 2018.

[127] B. Fitzgerald and S. Klaas-Jan, "Continuous Software Engineering and Beyond: Trends and Challenges," in *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering*, Hyderabad, India, 2014.

[128] C. Shamieh, Continuous Engineering for Dummies, IBM Limited Edition, John Wiley & Sons, Inc., 2014.

[129] The Apache Software Foundation, "Another Neat Tool (Ant)," [Online]. Available: http://ant.apache.org.

[130] The JUnit Team, "JUnit," [Online]. Available: http://junit.org/.

[131] Checkstyle, "Checkstyle," [Online]. Available: http://checkstyle.sourceforge.net/.

[132] Vlad Roubtsov, "EMMA," [Online]. Available: http://emma.sourceforge.net.

[133] University of Maryland, "FindBugs," [Online]. Available: http://findbugs.sourceforge.net/.

[134] DZone Research, "Continuous Delivery Maturity Checklist," *The DZone Guide to Continuous Delivery 2015 Edition,* p. 32, 2015.

[135] M. Loukides, What is DevOps, O'Reilly, 2012.

[136] AXELOS Ltd., "Information Technology Infrastructure Library (ITIL)," [Online]. Available: https://www.axelos.com/best-practice-solutions/itil.

[137] ISACA, "Control Objectives for Information and Related Technology (COBIT)," [Online]. Available: http://www.isaca.org/cobit/pages/default.aspx.

[138] ISO, "ISO/IEC 20000-1:2011 Information technology-Service management-Part 1: Service management system requirements," [Online]. Available: http://www.iso.org/iso/iso_catalogue/catalogue_ics/catalogue_detail_ics.htm?csnumber=51986.

[139] R. Lowery, "What You Need to Know about DevOps," 19 04 2016. [Online]. Available: http://www.solutionsiq.com/what-you-need-to-know-about-devops/.

[140] Derek Robert Price & Ximbiot, "Concurrent Versions Systems (CVS)," GNU, [Online]. Available: http://www.nongnu.org/cvs/.

[141] Perforce Software, Inc., "Perforce," [Online]. Available: https://www.perforce.com/.

[142] Microsoft Inc., "Team Foundation Server (TFS)," [Online]. Available: https://www.visualstudio.com/tfs/.

[143] J. Narebski, Mastering Git, Packt Publishing, 2016.

[144] A. Alipour, A. Hindle and S. Eleni, "A Contextual Approach Towards More Accurate Duplicate Bug Report Detection," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, San Francisco, CA, USA, 2013.

[145] J. C. Campbell, E. A. Santos and A. Hindle, "The Unreasonable Effectiveness of Traditional Information Retrieval in Crash Report Deduplication," in *Proceedings of the 13th International Conference on Mining Software Repositories*, Austin, Texas, 2016.

[146] J. C. Campbell and A. Hindle, "The charming code that error messages are talking about," *PeerJ PrePrints,* vol. 3, p. e1388, 2015.

[147] R. Southwick, "Hospital staffer fired for snooping into patient, employee health files," *Edmonton Journal,* p. A8, 08 10 2014.

[148] T. Hopper, "Mount Sinai staff access Ford's records - Two employees disciplined for taking a peek," *National Post,* p. 9, 17

10 2014.

[149] Security Standard Council, "Payment Card Industry (PCI) Payment Application Data Security Standard v1.1," 2008.

[150] C. Howson, J. Parenteau, R. L. Sallam, T. W. Oestreich, J. Tapadinhas and K. Schlegel, "Critical Capabilities for Business Intelligence and Analytics Platforms," Gartner, 2016.

[151] M. Zetlin, "How to balance maintenance and IT innovation," 21 10 2013. [Online]. Available: http://www.computerworld.com/article/2486278/it-management/how-to-balance-maintenance-and-it-innovation.html.

[152] A. Cahyadi, Beyond Functionality and A User Interface: A Design Thinking Perspective on The Design of Dashboards, vol. Ph.D. Thesis, Faculty of Business and Law, Swinburne University of Technology, 2016.

[153] O. Lasila, R. R. Swick and World Wide and Web Consortium, *Resource Description Framework (RDF) Model and Syntax Specification,* W3C Recommendation, 1998.

[154] R. T. Fielding, "Chapter 5 Representational State Transfer (REST), Architectural Styles and the Design of Network-based Software Architectures, Doctoral dissertation," University of California, Irvine, 2000. [Online]. Available: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.

[155] S. Dustdar and W. Schreiner, "Survey on Web services Composition," *International Journal on Web and Grid Services,* vol. 1, pp. 1-30, 2005.

[156] S. Ran, "A Model for Web Services Discovery With QoS," *ACM SIGecom Exchanges,* vol. 4, no. 1, pp. 1-10, Spring 2003.

[157] Z. Ives, N. Khandelwal, A. Kapur and M. Cakir, "ORCHESTRA: Rapid, Collaborative Sharing of Dynamic Data," in *The 2nd Biennial Conference on Innovative Data Systems Research (CIDR'05)*, Asilomar, CA, USA, 2005.

[158] M.-T. Schmidt, B. Hutchison, P. Lambros and R. Phippen, "The Enterprise Service Bus: Making servie-oriented architecture real," *IBM Systems Journal,* vol. 44, no. 4, pp. 781-797, 2005.

[159] A. Nigam and N. Caswell, "Business artifacts: An approach to operational specification," *IBM Systems Journal,* vol. 47, no. 3, pp. 428-445, 2003.

[160] R. Hull, "Artifact-Centric Business Process Models: Brief Survey of Research Results and Challenges," in *OTM 2008*, Monterrey, Mexico, 2008.

[161] W. Jones, "Personal Information Management," *Annual Review of Information Science and Technology,* vol. 41, no. 1, pp. 453-504, 2007.

[162] H. Ludwig, J. Laredo, K. Bhattacharya, L. Pasquale and B. Wassermann, "REST-Based Management of Loosely Coupled Services," in *The 18th international conference on World Wiide Web (WWW'09)*, Madrid, Spain, 2009.

[163] HL7, "Health Level Seven International," [Online]. Available: http://www.hl7.org.

[164] GS1 US, "RosettaNet Standards," [Online]. Available: https://resources.gs1us.org/RosettaNet.

[165] EDIFACT, "United Nations Directories for Electronic Data Interchange for Administration, Commerce and Transport," [Online]. Available: http://www.unece.org/trade/untdid/welcome.htm.

[166] I. Manotas, C. Bird, R. Zhang, D. Shepherd, C. Jaspan, C. Sadowski, L. Pollock and J. Clause, "An Empirical Study of Practitioners' Perspectives on Green Software Engineering," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, 2016.

[167] M. Zakary and L. Gillam, "Energy efficient computing, clusters, grids and clouds: A taxonomy and survey," *Sustainable Computing: Informatics and Systems,* vol. 14, pp. 13-33, June 2017.

[168] J. Scaramella and M. Eastwood, "Solutions for the Datacenter's Thermal Challenges," Jan 2007. [Online]. Available: http://whitepapers.zdnet.com/abstract.aspx?docid=352318.

[169] B. Göetz, T. Peierls, J. Bloch, J. Bowbeer, D. Holmes and D. Lea, Java concurrency in practice, Addison-Wesley, 2006.

[170] G. Pinto, F. Castor and D. Y. Liu, "Mining Questions About Software Energy Consumption," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, 2014.

[171] C. Pang, "Technical Summary: What do developers know about Software Energy consumption and power use?," 2013. [Online]. Available: http://webdocs.cs.ualberta.ca/~hindle1/2014/green-programmers/.

[172] A. Hindle, "Green Mining: A Methodology of Relating Software Change to Power Consumption," in *Proceedings of the 9th Working Conference on Mining Software Repositories*, 2012.

[173] P. Kurp, "Green computing," *Communications of the ACM,* vol. 51, no. 10, pp. 11-13, 2008.

[174] H. Khalid, E. Shihab, M. Nagappan and A. E. Hassan, "What Do Mobile App Users Complain About?," *IEEE Software,* vol. 32, no. 3, pp. 70-77, 2015.

[175] D. Li, S. Hao, J. Gui and W. Halfond, "An Empirical Study of the Energy Consumption of Android Applications," in *Proceedings of the IEEE International Conference on Software Maintenance and Evolution*, 2014.

[176] A. Banerjee, L. K. Chong, S. Chattopadhyay and A. Roychoudhury, "Detecting energy bugs and hotspots in mobile apps," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2014.

[177] C. Zhang, A. Hindle and D. M. Germán, "The Impact of User Choice on Energy Consumption," *IEEE Software,* vol. 31, no. 3, pp. 69-75, 2014.

[178] P. Naidu, "A DevSecOps Reference Guide 101," DZone, Inc., 27 07 2018. [Online]. Available: https://dzone.com/articles/a-devsecops-reference-guide-101. [Accessed 10 08 2018].

[179] V. Puranik and J. Worthington, "Scale DevSecOps with your Continuous Integration Pipeline," Veracode, 14 02 2019. [Online]. Available: https://webinars.devops.com/scale-devsecops-with-your-continuous-integration-pipeline.

[180] Threat Stack, Inc., "The SecOps Playbook - How SecOps Enables Secure Code Release, At Scale and At Speed," 2016.

[181] E. Chickowshi, "7 Reasons Why CISOs Should Care About DevSecOps," Security Boulevard, 2019.

Appendix A Interview Questions for Initial Raw Data Gathering

What do programmers know about Continuous Integration (CI) details?

Repository:

Do your CI processes involve source code repository software?

If so, what is/are the source code repository software?

Do you store anything else in the repository, besides source code?

Build script:

Do your CI processes involve tools that automate build from source code?

If so, what are the automated build tools?

Do your CI processes practice single command build?

Do you schedule periodic application builds on a dedicate machine?

If so, when a scheduled build fails, would the team be acknowledged?

If so, through what mechanism is the team notified?

Do you think it is costly to dedicate a machine just for build?

Development Integrated Development Environment (IDE):

Do you use IDEs?

If so, which IDEs?

Can you build your applications without the IDE?

Testing:

Do you execute performance or load tests?

Do your CI processes involve automated (regression) tests?

If so, do you use any automated test tools?

What are the automated test tools?

Are any of your regression tests based on specific authentication and authorization?

Are any of your regression tests based on specific data?

If so, how do you manage the test data?

Do any of your regression tests require communication with external web services?

Code Quality:

Do your CI processes involve source code inspection (analysis) tools?

If so, what are the inspection tools and what are being inspected?

Deployment:

Do your CI processes generate deployable software, such as installable packages?

Do your CI processes include automated deployment?

If so, how many different machines does your build process deploy to?

Appendix B Open Coding Result of the Initial Interviews

Inter view	Code	Related code	Context/Expression from Transcription
#1	programming experience	experience	Education is experience?
#1	education	training	University formal training regard as experience to student.
#1	employed		Employment implies professionalism?
#1	full stack		Both client and server sides.
#1	frequent		Frequency and continuous are related.
#1	integration problem		CI minimizes integration problems.
#1	merge		Merging is integration.
#1	fine granularity		Tones of changes
#1	Git	tool (repository)	Version control is a CI tool
#1	Git	version control	
#1	Git	tool (repository)	CI is Git?
#1	improve quality		CI improves quality regardless of team size in a single developer team.
#1	reduce repetitive		CI reduces repetitiveness.
#1	maintainability		CI improves maintainability.
#1	reduce defect		CI improves quality.
#1	familiar with Git	tool (repository)	Tool dependency.
#1	overhead	cost	Worthwhile even with overhead.
#1	worth while	ROI	
#1	overhead	cost	CI reduces maintenance overhead.
#1	cost		Ignore installation cost.
#1	emotional effect		Breaking CI is not a big deal.
#1	experience		Experience with Git.
#1	experience		On small or medium projects.
#1	repository	tool	CI needs version control.
#1	repository	tool	Just source code in repository.
#1	automation		Unclear coverage.
#1	knowledge	training	Lack of build system knowledge.
#1	knowledge	training	Lack of CI knowledge.
#1	auto test		CI without automated test?
#1	cooperation		Lack of cooperation or periodic build.
#1	dedicate machine		Cost of build machine is a concern.
#1	training		Programmer benefit from training or survey.
#1	project scope		CI benefits less obvious in small project.

Inter view	Code	Related code	Context/Expression from Transcription
#1	abstract concept		
#2	experience	education	Education and programming assignments experience?
#2	experience		Interesting question about years of experience.
#2	experience	education	Taking Java course, but never used, valid experience?
#2	experience		How to quantify experience?
#2	experience		Quantify usage of technical skill.
#2	experience		Hobbyist activity counts as experience, or must be production coding experience?
#2	experience	education	Years of school as experience?
#2	experience		More experienced programmers do not count school years as experience.
#2	experience		After working in industry, realized that schools provided limited knowledge.
#2	education	training	College training practical enough for work?
#2	repository		Shared source code submit to repository
#2	integration		Is merging.
#2	merge		Is integration.
#2	automation		CI is automation.
#2	test		CI involves testing.
#2	deployment		CI can include CD.
#2	different process	adaption	Inclusive or exclusive?
#2	partial adaption	adaption	
#2	tool		Tools are necessary in CI
#2	CircleCI	tool	CI tool.
#2	tool		Automate code submission, triggers build, test and deployment.
#2	repetitive		Part of programmer's life.
#2	repetitive		CI reduces repetitiveness.
#2	maintainability		With associated cost.
#2	configuration		Environment.
#2	self-documentation		CI processes are self-documented.
#2	scaled effect		CI has mixed positive and negative effects.
#2	partial adaption	adaption	
#2	maintainability		Related to CI.
#2	quantifiable benefit		CI’s benefit is hard to quantify.
#2	reduce defect		In final product
#2	final product		CI interim benefits are hard to quantify.
#2	final product		CI over all benefit is undoubtable.

Inter view	Code	Related code	Context/Expression from Transcription
#2	prevent naive error	reduce defect	
#2	multiple approach	experience	Experience leads to different approach.
#2	multiple approach	experience	Experience identifies complication.
#2	multiple approach		For different requirements.
#2	multiple approach		May add to work load.
#2	multiple option	multiple approach	
#2	easy approach	adaption	Start with basic and minimal CI processes.
#2	easy approach		No automated infrastructure maintenance.
#2	maturity		More mature CI brings more benefit?
#2	maturity		Experienced worker realize the need of multi-role knowledge.
#2	enforcement		Avoid inconsistence in human.
#2	consistency		CI enforces consistence.
#2	overhead	cost	Benefit cancels out the cost and overhead
#2	multi-environment	configuration	Identify additional defects.
#2	pay for better		Paid service is better?
#2	pay for better		Freeware creates problem or uncertainty?
#2	hardware support		Programmer lacks hardware knowledge for root cause analysis.
#2	multi-discipline	multi-role	CI needs multiple discipline knowledge during integration.
#2	multi-discipline	multi-role	Multiple discipline problem is hard to identify.
#2	social pressure	emotional effect	Some feel embarrassment
#2	social pressure	emotional effect	Programming is not a black room process anymore.
#2	social pressure	emotional effect	Programming is now a social event.
#2	quick return	ROI	Benefit can be recognized easily.
#2	code integration		CI starter.
#2	documentation	self-documentation	Part of CI content.
#2	build tool	tool	CI tool?
#2	single command build	build process	CI requirement?
#2	DevOps tool	DevOps	"ssh" deploy.
#2	DevOps tool	tool	
#2	build process		Not clearly defined.
#2	build process		What do include or exclude?
#2	perioditic build	auto build	
#2	feedback		Need feedback mechanism.
#2	IDE	tool	Commonly used
#2	IDE	tool	Used in development, testing, deployment?

Inter view	Code	Related code	Context/Expression from Transcription
#2	performance test		Need lot of effort to maintain, not being kept up
#2	GUI test tool	tool	Part of CI?
#2	role-based authorization		Missing test concept.
#2	test data management		Manually done.
#2	external web service access		Becoming popular.
#2	JSLint	tool	CI static analysis tool.
#2	static analysis tool	tool	Part of CI?
#2	static analysis tool	tool	Too many warnings.
#2	warning fatigue		Warning discarded.
#2	script	tool	Script is tool?
#2	packaging		In deployment.
#2	CD		Part of CI?
#2	common understanding	standard	Lack of common understanding among IT practitioners.
#2	training		Necessary in CI.
#2	CI survey	knowledge	IT practitioner benefits with knowledge.
#3	version control		Confuse or conflate CI with version control.
#3	code integration		Merge weeds out conflicts.
#3	keep history		Need audit data in development.
#3	accountability		Take responsibility.
#3	improve quality		Cannot quantify
#3	repetitive		Reduce repeating merge, build, and test.
#3	reduce defect		In term of removing conflict.
#3	feedback		Timely feedback for developers.
#3	IDE	tool	Tool dependence.
#3	IDE	tool	Not separable from CI.
#3	tool dependence	tool	
#3	auto build	build process	Less known area in CI.
#3	database integration		Less known area in CI.
#3	energy efficiency		Lack of knowledge about software energy consumption.
#3	patch management		Concept rarely mentioned in CI.
#3	role-based authorization		Multiple authorization not considered.

Inter view	Code	Related code	Context/Expression from Transcription
#3	test data management		Not considered.
#3	experience		CI depends on individual's experience.
#3	expenses	cost	Return on investment (ROI)
#3	auto build	build process	Not used.
#3	auto build		
#3	collaboration	social pressure	Lack of or avoided.
#3	dynamic type language		Not integrate well with CI.
#3	dynamic type language	auto build	Not compliable, not buildable.
#3	dynamic type language	auto test	Need automated test.
#3	professional development	training	Lack of.
#3	immuture developer	experience	
#3	immuture developer	social pressure	
#4	merge		Merging is integration?
#4	feedback	test	Unit tests provide feedback.
#4	feedback		Promptly feedback, most important in CI.
#4	Maven	tool	CI tool.
#4	Maven	auto build	
#4	automation		Automation is CI.
#4	automation		Limitation of automation?
#4	automation		Human involvement in automation?
#4	automation		Quantify different levels?
#4	readiness		Always ready.
#4	readiness		Quantify readiness?
#4	continuous test	auto test	
#4	improve quality		In feeling or actual better?
#4	improve quality	feedback	By feedback?
#4	instantaneous feedback	feedback	
#4	adaption		Practitioners are convinced by CI.
#4	manual process		Build, run, unit test.
#4	reduce repetitive		What manual process reduced?
#4	unit test	test	More than unit test in CI.
#4	feedback	improve quality	Quantify improved quality?
#4	reduce defect	tool	By supporting tools.
#4	reduce defect	tool	Closely related to tool usage.

Inter view	Code	Related code	Context/Expression from Transcription
#4	reduce defect	tool	Making tools meaningful.
#4	reduce defect	tool	Tools make reducing defect feasible?
#4	peer review		Manual process.
#4	process	improve quality	Improve quality?
#4	auto build		Fundamental in CI.
#4	code integration		Fundamental in CI.
#4	repository		Fundamental in CI.
#4	technical debt		Reduce accumulative defect.
#4	maintenance overhead	overhead	Reduce maintenance overhead.
#4	maintenance overhead	cost	
#4	early discovery		Reduce maintenance overhead.
#4	learning curve	training	CI education necessary.
#4	progressive adaption	adaption	Quantify different adaption levels.
#4	adaption level	adaption	Different levels of CI.
#4	performance test		Cloud solves scaling problem.
#4	performance test		No need for scaling test?
#4	cloud	infrastructure	
#4	container		CI platform.
#4	Docker	container	Enhance CI
#4	automation		Extend to platform configuration.
#4	container		Simplify CI processes.
#4	container		Improve deployment process.
#4	Amazon	cloud	
#4	AWS	cloud	
#4	DevOps		Closely related to CI.
#4	configuration		CI extends to configuration.
#4	infrastructure		CI extends to infrastructure.
#4	monitor tool	tool	Necessary for CI.
#4	container	continuous test	Improve continuous test.
#4	dashboard	feedback	Method of reporting.
#4	continuous maintenance	maintainability	Rarely considered processes.
#4	quality assurance	improve quality	Repetitiveness guarantees repeatable result.
#4	quality assurance		Learned the auditing and compliance lesson the hard way.
#4	continuous maintenance	maintainability	System and infrastructure administration, e.g. backup.
#4	continuous maintenance	maintainability	Currently not done.

Inter view	Code	Related code	Context/Expression from Transcription
#4	third-party service		GitHub
#4	repository		Pros and cons between local and remote installation.
#4	privacy		Code ownership as property.
#4	third-party service	cloud	
#4	repository		Backup important for local repository.
#4	automation		Involve manual process in CI.
#4	Agile		Roles in CI?
#4	multi-role		Multiple hats.
#4	availability	third-party service	CI depends on third-party service availability.
#4	version control	tool (repository)	Part of CI
#4	repository	tool	Different repository software simulates different behavior.
#4	packaging		For distribution.
#4	version control		Important in CI.
#4	build		Important in CI.
#4	rapid change		Technology needs to catch up.
#4	AWS	cloud	
#4	service versioning		Multi-version service support is challenging in CI.
#4	service versioning		Deploy different versions of API.
#4	multi-client		Induce multi-version application.
#4	multi-client	maintainability	Add maintenance challenges.
#4	data as a service		
#4	data maintenance	maintainability	Important in CI
#4	political limitation		Country boundary restrains deployment.
#4	legislation	cloud	Limit cloud data usage.
#4	privacy	cloud	Limit cloud data usage.
#4	auto GUI test	GUI test tool	GUI test difficult to automate.
#4	GUI test tool		Revolution.
#4	auto GUI test	GUI test tool	Capture-based GUI test not practical.
#4	auto GUI test		Multi-platform issues.
#4	multi-platform	configuration	
#4	multi-platform	multi-environment	
#4	Capistrano	tool	Build tool.
#4	product line	multi-platform	Multi-platform issues.
#4	web technology		Easier to learn for startup than native app.
#4	startup		
#4	Kerberos		Security standard.
#4	auto test		Authentication and authorization challenges.

Inter view	Code	Related code	Context/Expression from Transcription
#4	security		Continuous security not effective?
#4	security		Big-bang more effective?
#4	privacy	test data management	Concern in test data management.
#4	regression test	performance test	Stress test.
#4	regression test	performance test	Important and challenging in CI.
#4	script	tool	Used for automation.
#4	fake Agile	Agile	Team claims to use Agile, but not.
#4	essential	startup	CI is not consider essential to most startup.
#4	cost		Monetary or non-monetary.
#4	cost		Resource.
#4	cost		People? Hardware? Software? Procedure?
#4	cost		What is needed to start CI?
#4	cost	startup	Startup cost.
#4	return on investment (ROI)	cost	Not enough to motivate CI adaption?
#4	process		Lack of excitement.
#4	phase two		Adapt CI in source code restructure.
#4	phase two		Adapt CI as secondary thought.
#4	phase two		Retrofit CI into existing structure.
#4	standard		Lacking in CI (e.g. Docker).
#4	craftsmanship		CI implementation depends on practitioner's skill set and experience.
#4	single command build	standard	Need definition for single command build.
#4	Rubymine	tool (IDE)	
#4	JetBrain	tool (IDE)	
#4	SourceLab	tool	CI verification tool.
#4	SE skill		Programming language.
#4	SE skill	training	Proficient in programming not sufficient for CI.
#4	SE skill	knowledge	Practitioners need end-to-end knowledge to support CI.
#4	Jack of all trades	DevOps	CI needs wide range of skill.
#4	size matter		CI culture different between large and small organizations.
#4	size matter		Large organizations have more divided responsibilities.
#4	size matter		Small organizations have no divided responsibilities.

Table B.1 Codes of the initial interviews

Appendix C Open Coding Trees

1st Level	2nd Level	3rd Level	4th Level
experience			
	employed		
	programming experience		
		immature developer	
	abstract concept		
	training		
		education	
		professional development	
		learning curve	
	knowledge		
		full stack	
		multiple approach	
		multi-role	
		multi-discipline	
		CI survey	
		SE skill	
		Jack of all trades	
		DevOps	
	craftsmanship		

Table C.1 Code Tree for "Experience"

1st Level	2nd Level	3rd Level	4th Level
practice			
	frequent		
	consistency		
	merge		
	code integration		
		database integration	
	automation		
	test		
		auto test	
		dedicate machine	
		performance test	
			regression test
		role-based authorization	
		test data management	

1st Level	2nd Level	3rd Level	4th Level
		energy efficiency	
		test data management	
		dynamic type language	
		continuous test	
			container
		unit test	
	deployment		
		continuous deployment	
	process		
		different process	
		manual process	
			peer review
	configuration		
		multi-environment	
		multi-platform	
		product line	
	multiple approach		
		multiple option	
	documentation		
		self-documentation	
	build		
		auto build	
		build process	
		single command build	
		periodic build	
		dynamic type language	
	DevOps		
	feedback		
		instantaneous feedback	
	packaging		
	patch management		
	continuous maintenance		
	security		
		privacy	
		test data management	
	data maintenance		

Table C.2 Code Tree for "Practice"

1st Level	2nd Level	3rd Level	4th Level
benefit			
	integration problem		
		pay for better	
	version control		
		keep history	
		service versioning	
	fine granularity		
	improve quality		
		feedback	
		process	
		quality assurance	
	reduce repetitive		
		repetitive	
	maintainability		
		multi-client	
	reduce defect		
		prevent native error	
		tool	
	self-documentation		
	quantifiable benefit		
		final product	
	accountability		
	readiness		
	technical debt		
	early discovery		
	availability		
		third-party service	
	rapid change		

Table C.3 Code Tree for "Benefit"

1st Level	2nd Level	3rd Level	4th Level
cost			
	ROI		
		quick return	
	adaption		
		partial adaption	
		easy approach	
		maturity	
		scaled effect	
		progressive adaption	

1st Level	2nd Level	3rd Level	4th Level
		adaption level	
	overhead		
		worth while	
		maintenance	
	expenses		

Table C.4 Code Tree for "Cost"

1st Level	2nd Level	3rd Level	4th Level
tool			
	repository		
		Git	
	build		
		CircleCI	
		Maven	
	DevOps		
	IDE		
		Rubymine	
		JetBrain	
	test		
		GUI test	
			SourceLab
	static analysis		
		JSLint	
	script		
	tool dependence		
	infrastructure		
		hardware support	
		cloud	
			Amazon (AWS)
		container	
			Docker
	monitor		
	feedback		
		dashboard	
	security		
		Kerberos	
	deployment		
		Capistrano	

Table C.5 Code Tree for "Tool"

1st Level	**2nd Level**	**3rd Level**	**4th Level**
social aspect			
	collaboration		
	cooperation		
	project scope		
		Agile	
		fake Agile	
	enforcement		
	social pressure		
		emotional effect	
	warning fatigue		
	standard		
		common understanding	
	political limitation		
		legislation	
	startup		
		essential	
		phase two	
	size matter		

Table C.6 Code Tree for "Social Aspect"

1st Level	**2nd Level**	**3rd Level**	**4th Level**
third-party service			
	cloud		
		Amazon (AWS)	
	external web service access		
	data as a service		
	web technology		

Table C.7 Code Tree for "Third-Party Service"

Appendix D Potential DevOps Research Questions

Experience

- Knowledge across multiple fields is necessary to practice CI?
- Need to improve CI education?
- CI depends heavily on individual's experience?
- Does the level of CI adaption in a company correspond to the experience of its IT practitioner?
- Does CI demand all IT practitioners to have some programming ability?
- Does CI demand higher skill set from IT practitioners?
- How much infrastructure knowledge IT practitioner should have to practice CI?
- What are the skills and knowledge needed for CI?
- Is CI covered in education curriculum?
- Is CI a topic in recent Software Engineering researches?
- What and where do CI specialists get their training?
- What are the requirements for CI jobs?
- How popular are CI positions in job posting sites?
- What should be the qualification of CI specialist?
- Is there enough publicity about CI?
- How do IT practitioners learn CI? Junior? Senior?
- Is CI pure craftsmanship?
- How can Software Engineering education cover the CI needs?
- CI experts need to be Jack of all trades?

Practice

- What are the CI maintenance considerations?
- How is practicing CI different between large and small teams?
- Is CI a practice for single application?
- Maintenance is dead?
- CI includes "Continuous" and "Integration", but the "Continuous" concept is always overlooked by the practitioners?
- Does adapting CI mean eliminating scheduled releases and milestones?
- How using different version control repositories are affecting CI practices?
- What be managed by CI? Configuration? Environment? Virtual machines?
- How enterprises handle broken builds?
- How enterprises manage CI infrastructure?
- What are the common concerns for CI environment and environment management?
- How applications depending on third-party services are tested and managed by CI?
- How often is Continuous Inspection adopted?
- Why are IT practitioners confusing version control repository with CI?
- How to manage mobile application patching in CI?

- Does CI work for dynamic typed languages, such as Python?
- What tasks should be included in the application build process?
- How to manage secondary artifacts (e.g. scripts) in CI?
- How does CI affect the IT practices?
- What are the feedbacks provided through CI?
- What are the major causes of CI build failures for different types of applications?
- What is hindering people from adapting auto-test?
- How CI changes software maintenance?
- CI practices and automation intensify the usage of many resources, and escalates maintenance issues which have been ignored?
- What are the issues raised from adopting CI?
- How companies adopt CI?
  What are the CI adaptation processes?
- Are CI and DevOps separable?
- In CI, who should be responsible for cleanup, archive and backup?
- How can CI practices be incorporated into project planning and management?
- Are there specific technologies that do not integrate well with CI (e.g. Perl)?
- Can human involved GUI testing be replaced by CI?
- What are included in software security testing?
  How often is security testing included in CI?
- Comparing continuous security review versus big-bang security review?
- How to evaluate the security and privacy quality of an application in the CI process?
- Is there CI security evaluation?
- How are CI practices different between large and small companies?
- It has been a challenge to manage mutli-version service APIs.
  How often is CI used in this type of environment?

Benefit

- What are the CI's benefits?
  What CI is benefiting?
- Can CI's benefits be quantified?
- Have CI's benefits been proven?
- Are there any interim benefits until CI is fully implemented?
- Has it been proven that CI reduces defects?
- Does implementing CI improve quality?
- Without using tools, can CI improve quality and reduce defects?
- Does feedback from the CI processes improve software quality?
- How does feedback improve quality?
- Does the scale of return from implementing CI correspond to adaption level, team size, and/or application size?
- What kinds of defects CI can catch, while compiler cannot?
- CI enforces consistency therefore reduce human errors. Does it reduce defects?
- Is there data that compare productivity before, during and after CI adoption?

- What are the quick wins from CI?
- What are the benefits correspond to each CI process?
- What are the most important goals in adopting CI?
- What are the defects CI reduced?
- Can automated testing's benefit be qualified?
- CI is more beneficial to complicated projects and applications?
- Does CI make application security better, worse, or irrelevant?
- CI improves awareness by feedback?

Cost

- What is the total cost of ownership for CI? Man hours? Software? Hardware?
- What are the hard and soft expenses in adopting CI?
- How can ROI be calculated for adopting CI?
- How to estimate and evaluate CI adoption costs?
- How can startups plan for CI costs?

Tool

- Engineers do everything by the book. Is CI that book for Software Engineering?
- CI tools usually run on one platform, not good for applications using or support multiple platforms?
- Will enterprises use open source CI tools? For development? For production?
- What can be accomplished in CI without using third party tools?
- Comparing startup and enterprise, how valuable are CI tools?
- What are the CI test data managing processes and tools?
- How different IDEs support different CI processes?
- Are CI tools integrating well?
- How often company integrates container usage into CI?
- How the container technology affects CI?
- What are the differences between using local versus remote repository for CI?
- Any suggestion or guidelines for selecting repository for CI?
- Is there CI tool that build, test and deploy for multiple environments?
- Is there data management tools that support CI?
- All CI tools must support command-line interface?
- How are CI processes different between the bare metal, virtual machines, and container environments?
- What programming languages are used for CI automation?

Social Aspect

- How CI affects the IT culture?
- Is coordination really necessary in CI?
  If so, what kind of coordination is necessary in CI?

- What are the roles and responsibilities of IT support personnel in CI?
- Has CI killed maintenance?
- Does causing a build failure have any emotional effect on the IT practitioners?
- Does CI affect team morale?
- How to promote CI to non IT practitioners?
- Has CI created negative social effect?
- Who should lead CI adoption (e.g. programmer, operator, business manager)?
- Does professional designation have any value in IC?
- How programmers with different style adopt CI from the social aspect?
- How CI handle warning fatigue in programming?
- Is CI really a job killer?
- Are IT practitioners feeling good about CI?
- How does Agile work with CI?
- How much CI should startups adopt?
- How do companies manage personnel during CI adoption?

Third-Party Service

- What are the challenges of CI with cloud?
- What are the considerations of using third party services for CI?

Appendix E Potential Grounded Theory

Title: Continuous Integration (CI) is a craftsmanship. How do you learn the craft?

The Oxford dictionary defines craftsmanship as skill in a particular craft. The quality of design and work is shown in something made by hand artistically. Wikipedia describes craftsmanship as a human attribute relating to knowledge and skill at performing a task. Dictionary.com portrays a craftsman as a person who practices or is highly skilled in a craft.

CI is a craftsmanship for it is a kind of art that highly depends on the skill of the crafter. CI solutions are customized one at a time. They cannot be mechanically reproduced. IT practitioners craft CI solutions with their advance knowledge and sophisticated skill one at a time.

Art products may vary significantly, or be similar but different. Some crafted arts may have similar pattern, but created by different tools. Like crafting a piece of wood, there is no right or wrong. A crafter employs one's own skill and tools to create the best piece of art from the wood. Similarly, the implementation of CI is different from project to project, and from organization to organization. The CI implementation in large organizations may be different from that in small organizations. IT practitioners use their skills and tools to customize CI solution for the best of each situation. Since IT practitioners may have different skills and tools, similar solutions may employ different tools. Sometimes, CI strategy may be oriented according to the tools in hand. As the idiom says, "To a man with a hammer, everything looks like a nail."

Compared with other IT skills, CI demands higher craftsmanship. For CI solution involves the full IT stack, instead of one dimension, layer or component. Therefore, CI has to deal with a lot more variances.

Since CI is a craftsmanship, learning CI is different from learning other IT skills, for example, learning a programming language. To learn a programming language, one will learn its syntax, its data structure, its compiling and execution mechanism, etc.. However, a programming language is just one of the tools employed by CI. A student may master the skill of using hammer and chisel, but it does not means the student can engrave a good piece of art. Especially when CI is a full-stack art, mastering one skill is not sufficient to master all of CI. IT practitioners can learn their skills and tools through education, but they can only accumulate knowledge about how to apply these skills and tools in CI through hands-on experience. Just like crafting, students can only achieve proficiency in CI through hands-on experience.

Appendix F 2017 QS World University Rankings by Subject - Engineering and Technology

https://www.topuniversities.com/university-rankings/university-subject-rankings/2017/engineering-technology

Rank	University	Website	Country
1	Massachusetts Institute of Technology (MIT)	http://web.mit.edu/	USA
2	Stanford University	https://www.stanford.edu/	USA
3	University of Cambridge	https://www.cam.ac.uk/	UK
4	Nanyang Technological University (NTU)	http://www.ntu.edu.sg	Singapore
5	ETH Zurich - Swiss Federal Institute of Technology	https://www.ethz.ch/en.html	Switzerland
6	Imperial College London	https://www.imperial.ac.uk	UK
7	National University of Singapore (NUS)	http://www.nus.edu.sg/	Singapore
8	University of California, Berkeley (UCB)	http://www.berkeley.edu/	USA
9	University of Oxford	http://www.ox.ac.uk/	UK
10	Tsinghua University	http://www.tsinghua.edu.cn/publish/newthuen/index.html	China
11	The University of Tokyo	http://www.u-tokyo.ac.jp/en/	Japan
12	Ecole Polytechnique Federale de Lausanne (EPFL)	https://www.epfl.ch/	Switzerland
13	Harvard University	http://www.harvard.edu/	USA
14	Korea Advanced Institute of Science & Technology (KAIST)	http://www.kaist.edu/html/en/index.html	South Korea
15	The Hong Kong University of Science and Technology (HKUST)	http://www.ust.hk	Hong Kong, China
16	Georgia Institute of Technology	http://www.gatech.edu/	USA
17	California Institute of Technology (Caltech)	http://www.caltech.edu/	USA
=18	Peking University	http://english.pku.edu.cn/	China
=18	Tokyo Institute of Technology	http://www.titech.ac.jp/english/	Japan
20	Delft University of Technology	https://www.tudelft.nl/en/	Netherlands
21	Seoul National University	https://www.useoul.edu/	South Korea
22	Kyoto University	http://www.kyoto-u.ac.jp/en	Japan

Rank	University	Website	Country
23	National Taiwan University (NTU)	http://www.ntu.edu.tw/	Taiwan
=24	Politecnico di Milano	https://www.polimi.it/en/home/	Italy
=24	Shanghai Jiao Tong University	http://en.sjtu.edu.cn/	China
=24	Technical University of Munich	https://www.tum.de/en/homepage/	Germany
27	The University of Hong Kong	http://www.hku.hk/	Hong Kong, China
28	The University of Melbourne	http://www.unimelb.edu.au	Australia
29	KTH Royal Institute of Technology	https://www.kth.se/en	Sweden
30	University of California, Los Angeles (UCLA)	http://www.ucla.edu/	USA
=31	Carnegie Mellon University	http://www.cmu.edu/	USA
=31	The University of New South Wales (UNSW Sydney)	https://www.unsw.edu.au/	Australia
33	RWTH Aachen University	http://www.rwth-aachen.de/	Germany
34	University of Toronto	https://www.utoronto.ca/	Canada
=35	Technische Universitat Berlin (TU Berlin)	http://www.tu-berlin.de/menue/home/parameter/en/	Germany
=35	University Malaya (UM)	https://www.um.edu.my/	Malaysia
37	Princeton University	https://www.princeton.edu/	USA
=38	KIT, Karlsruhe Institute of Technology	https://www.kit.edu/english/	Germany
=38	National Tsing Hua University	http://nthu-en.web.nthu.edu.tw/bin/home.php	Taiwan
=38	University of Illinois at Urbana-Champaign	http://illinois.edu/	USA
41	The University of Sydney	http://sydney.edu.au/	Australia
42	Zhejiang University	http://www.zju.edu.cn/english/	China
43	Technical University of Denmark	http://www.dtu.dk/english	Denmark
=44	Monash University	https://www.monash.edu/	Australia
=44	University of Texas at Austin	https://www.utexas.edu/	USA
46	The Australian National University	http://www.anu.edu.au/	Australia
=47	Cornell University	https://www.cornell.edu/	USA
=47	Fudan University	www.fudan.edu.cn/en/	China
49	University of Michigan	https://umich.edu/	USA
50	The Chinese University of Hong Kong (CUHK)	http://www.cuhk.edu.hk/english/index.html	Hong Kong, China

Table F.1 Top 50 Institutes in 2017 QS World University Rankings by Subject - Engineering and Technology

Appendix G Academic Survey Invitation Letter

Subject: Hands-on experience between Engineering and Computer Science educators

Dear [Chair or Head] of [Engineering or Computer Science],

The work in Engineering and Computer Science both involve a lot of craftsmanship.
The skill of the craftsmen in their trade is highly dependent on their hands-on experience.
We wonder whether there is a different between Engineering and Computer Science undergraduate (Bachelor degree) educators in their hands-on experience.

We invite the top 50 universities in the 2017 QS World Rankings to help answering the question.
https://www.topuniversities.com/university-rankings/university-subject-rankings/2017/engineering-technology

Would you please kindly forward this invitation email to the [Engineering or Computer Science] educators in your [Engineering or Computer Science] department?

We invite full-time employees of post-secondary institutions with undergraduate teaching responsibilities (professor, associate professor, assistant professor, lecturer, etc.) in Engineering and Computer Science to answering these questions, which take just a minute.

1. Are you an Engineering or Computer Science educator?
2. How many years you have been teaching undergraduate Engineering or Computer Science courses?
3. How many years of full-time hands-on experience as an engineer or IT practitioner do you have?
4. Do you teach students about automated unit testing in your undergraduate courses?
5. Do you teach students about Continuous Integration (CI) in your undergraduate courses?
6. Do you teach students about DevOps in your undergraduate courses?

Link to the survey.
https://docs.google.com/forms/d/e/1FAIpQLScZ1eSW9eOUHdTuq3ixOnTBfptZSx4jzqN5FoR8B3LnMt0IMA/viewform

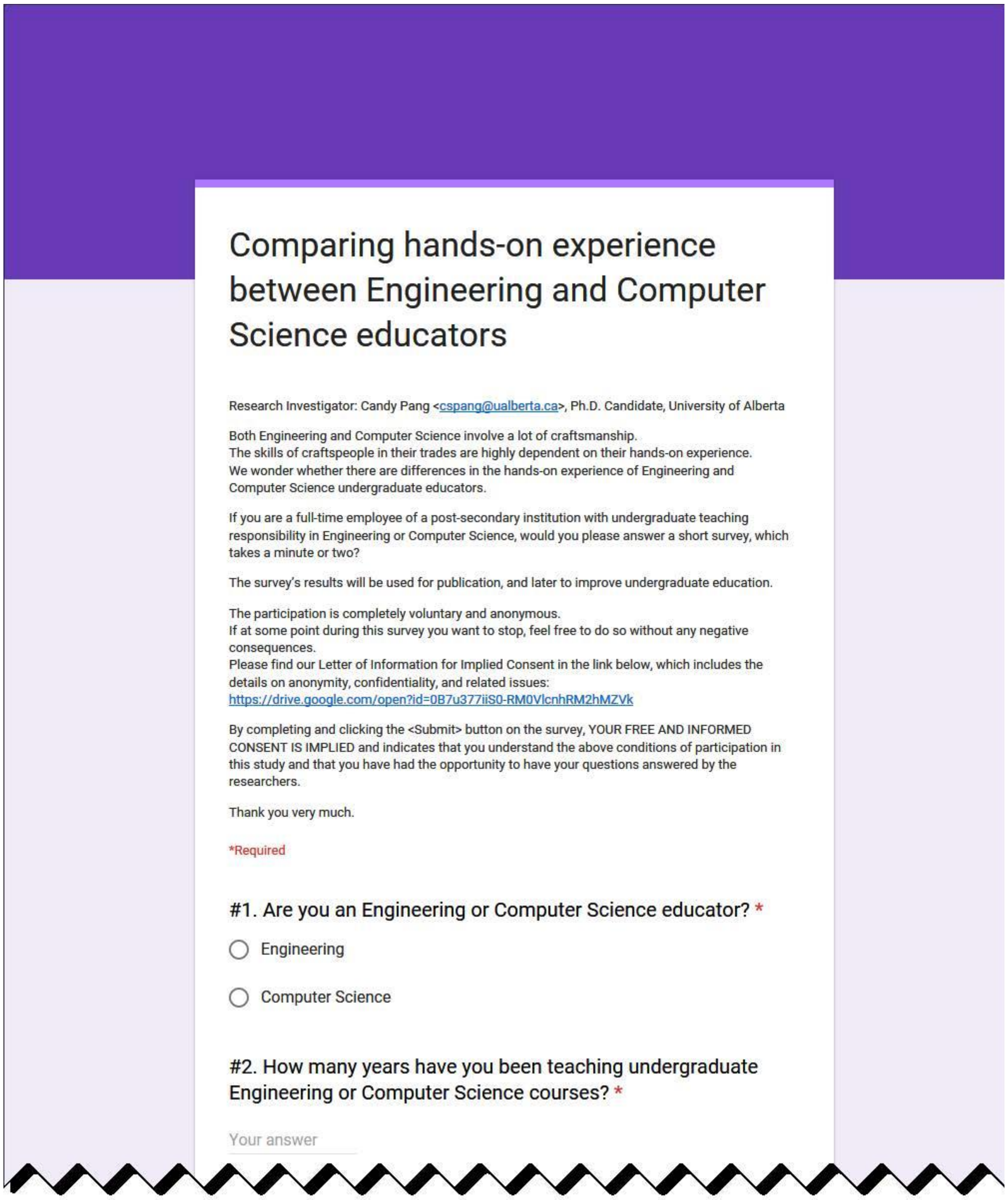
If you have any question, please feel free to email me.
Thank you very much.

Best regards,
Candy Pang
Ph.D. Candidate
University of Alberta

Email: cspang@ualberta.ca
Home page: http://webdocs.cs.ualberta.ca/~cspang/

P.S. The Letter of Information and Consent Form can be found at
https://drive.google.com/file/d/0B7u377iiS0-RdVgxbFFBN0U4M0U

Appendix H Survey comparing hands-on experience between Engineering and Computer Science educators



Comparing hands-on experience between Engineering and Computer Science educators

Research Investigator: Candy Pang <cspang@ualberta.ca>, Ph.D. Candidate, University of Alberta

Both Engineering and Computer Science involve a lot of craftsmanship.
The skills of craftspeople in their trades are highly dependent on their hands-on experience.
We wonder whether there are differences in the hands-on experience of Engineering and Computer Science undergraduate educators.

If you are a full-time employee of a post-secondary institution with undergraduate teaching responsibility in Engineering or Computer Science, would you please answer a short survey, which takes a minute or two?

The survey's results will be used for publication, and later to improve undergraduate education.

The participation is completely voluntary and anonymous.
If at some point during this survey you want to stop, feel free to do so without any negative consequences.
Please find our Letter of Information for Implied Consent in the link below, which includes the details on anonymity, confidentiality, and related issues:
https://drive.google.com/open?id=0B7u377iiS0-RM0VlcnhRM2hMZVk

By completing and clicking the <Submit> button on the survey, YOUR FREE AND INFORMED CONSENT IS IMPLIED and indicates that you understand the above conditions of participation in this study and that you have had the opportunity to have your questions answered by the researchers.

Thank you very much.

*Required

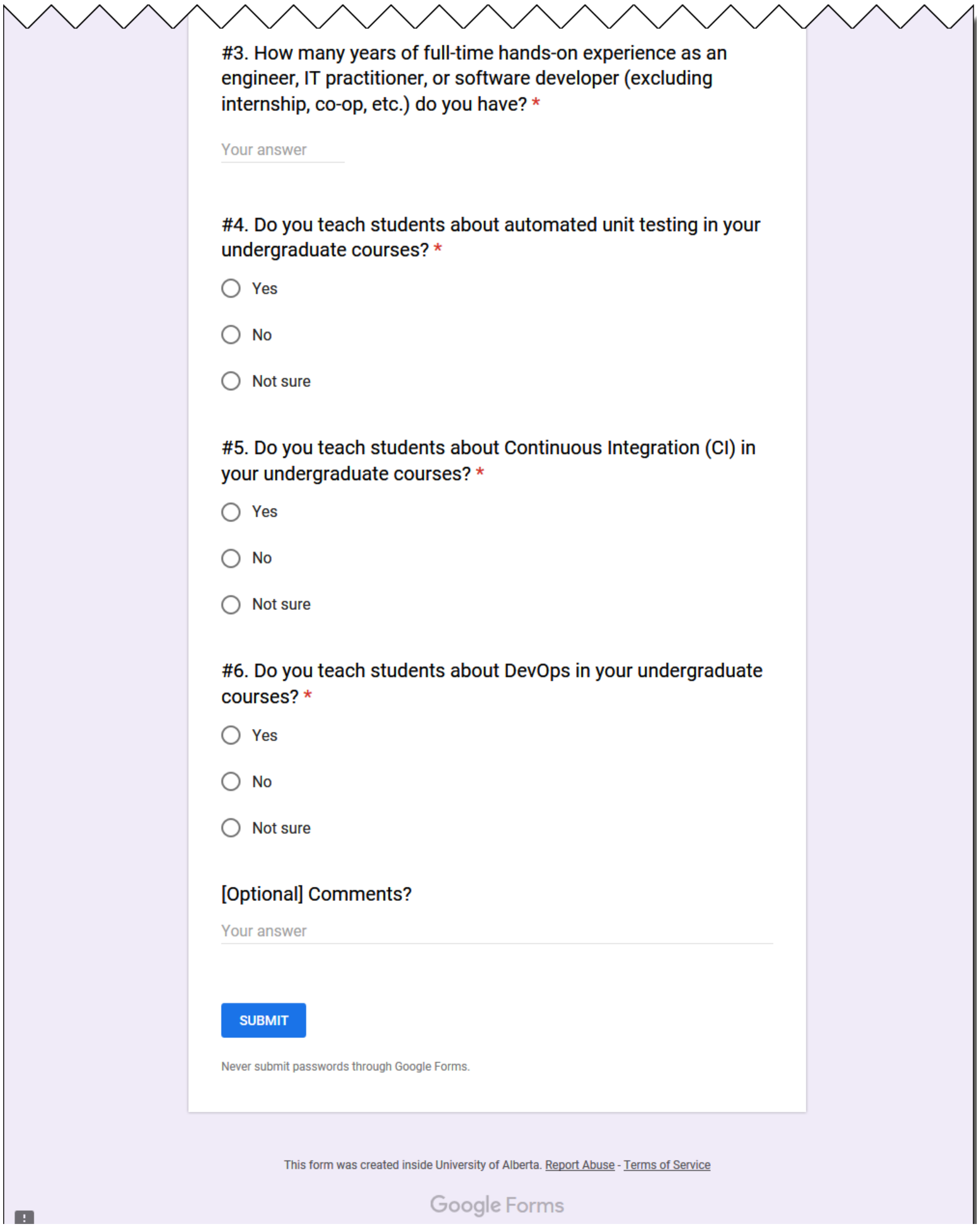
#1. Are you an Engineering or Computer Science educator? *

○ Engineering

○ Computer Science

#2. How many years have you been teaching undergraduate Engineering or Computer Science courses? *

Your answer



#3. How many years of full-time hands-on experience as an engineer, IT practitioner, or software developer (excluding internship, co-op, etc.) do you have? *

Your answer

#4. Do you teach students about automated unit testing in your undergraduate courses? *

- ◯ Yes
- ◯ No
- ◯ Not sure

#5. Do you teach students about Continuous Integration (CI) in your undergraduate courses? *

- ◯ Yes
- ◯ No
- ◯ Not sure

#6. Do you teach students about DevOps in your undergraduate courses? *

- ◯ Yes
- ◯ No
- ◯ Not sure

[Optional] Comments?

Your answer

Never submit passwords through Google Forms.

This form was created inside University of Alberta. Report Abuse - Terms of Service

Google Forms

Figure H.1 Comparing Hands-on Experience between Engineering and Computer Science Educators Survey

Appendix I Graduate Student Research Invitation Letter

Subject: What is Continuous Integration (CI)?

Dear fellow students,

Continuous Integration (CI), as part of Development-Operations (DevOps), is very popular in the industry.
Two new job titles "Automation architect" and "DevOps manager/VP of DevOps" are related to CI and DevOps, which "experts say may sound a bit strange but are needed in today's workplace and are here to stay." [1]
Research shows that "DevOps practices lead to higher IT performance. This higher performance delivers improved business outcomes, as measured by productivity, profitability, and market share." [2]

However, a survey of over 1000 IT practitioners showed that "around 60% either didn't know what [DevOps] was, or weren't sure if their companies were doing it." [3][4]

If you have not used CI or DevOps before, we are seeking your help in our research.
We will like to know what hinders people from adopting CI.
If you can give us half an hour, we will give you a 10 minutes introduction about CI.
Then, we will ask a few questions about your opinion of CI.
Please send me an email <cspang@ualberta.ca>, if you can give us half an hour.

If you have any question, please feel free to email me.
Thank you very much.

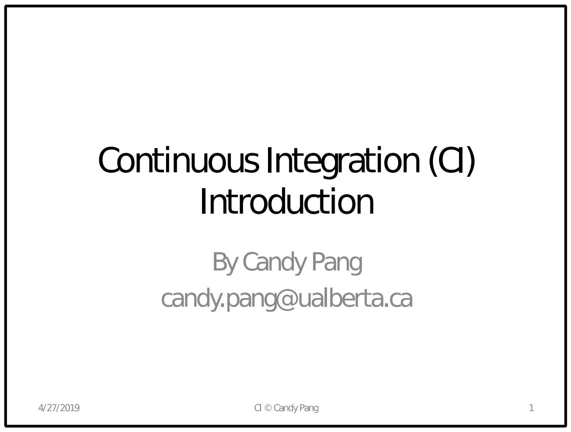
Best regards,
Candy Pang
Ph.D. Candidate
University of Alberta

Email: cspang@ualberta.ca
Home page: http://webdocs.cs.ualberta.ca/~cspang/

P.S. The Letter of Information and Consent Form can be found at
https://drive.google.com/open?id=0B7u377iiS0-RVXc3NVNmNlY2djg

[1] http://www.cio.com/article/3214629/careers-staffing/7-hot-new-it-jobs-and-why-they-just-might-stick.html
[2] https://puppet.com/system/files/2017-06/2017-state-of-devops-report_3.pdf
[3] https://dzone.com/articles/the-future-of-devops-2
[4] http://info.xmatters.com/rs/178-CPU-592/images/atlassian_devops_survey.pdf

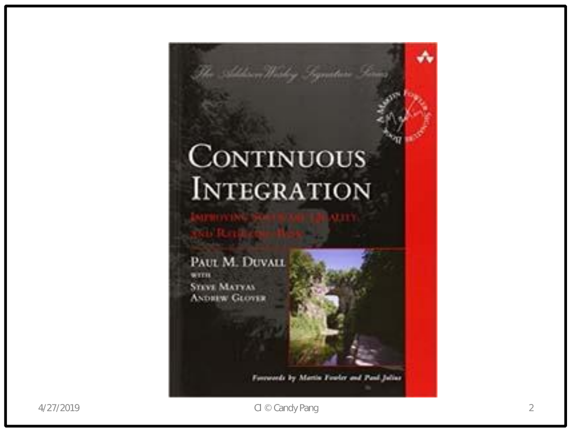
Appendix J Continuous Integration (CI) Introduction for Graduate Students



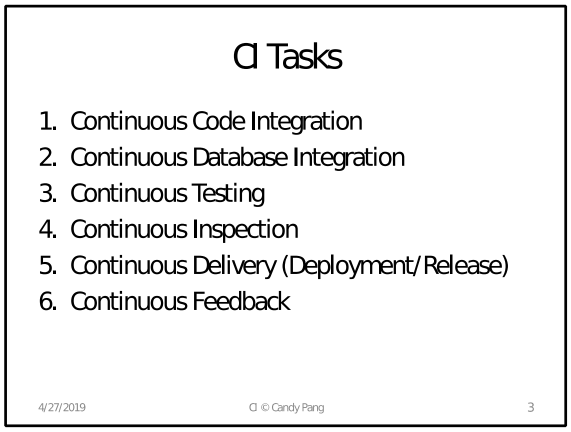
Continuous Integration (CI) Introduction

By Candy Pang
candy.pang@ualberta.ca

4/27/2019 CI © Candy Pang 1



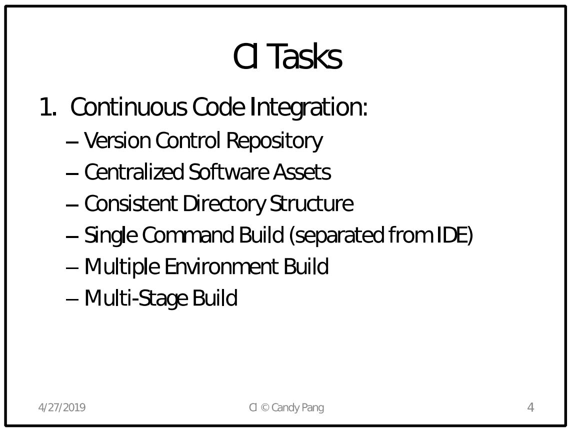
4/27/2019 CI © Candy Pang 2



CI Tasks

1. Continuous Code Integration
2. Continuous Database Integration
3. Continuous Testing
4. Continuous Inspection
5. Continuous Delivery (Deployment/Release)
6. Continuous Feedback

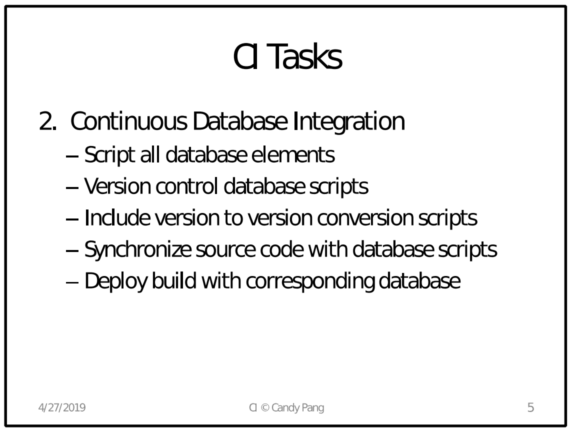
4/27/2019 CI © Candy Pang 3



CI Tasks

1. Continuous Code Integration:
   – Version Control Repository
   – Centralized Software Assets
   – Consistent Directory Structure
   – Single Command Build (separated from IDE)
   – Multiple Environment Build
   – Multi-Stage Build

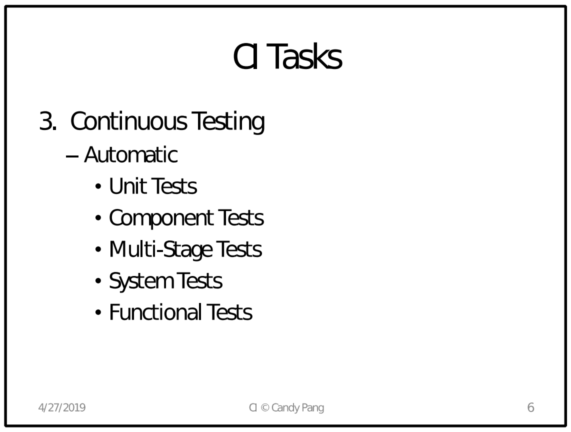
4/27/2019 CI © Candy Pang 4



CI Tasks

2. Continuous Database Integration
   – Script all database elements
   – Version control database scripts
   – Include version to version conversion scripts
   – Synchronize source code with database scripts
   – Deploy build with corresponding database

4/27/2019 CI © Candy Pang 5



CI Tasks

3. Continuous Testing
   – Automatic
     • Unit Tests
     • Component Tests
     • Multi-Stage Tests
     • System Tests
     • Functional Tests

4/27/2019 CI © Candy Pang 6

CI Tasks

3. Continuous Testing
   - Manual Test
     - Domain Experts Scripted Test
     - Smoke Test
     - Risk-Based Test
   - Performance Test
     - Response Time
     - Energy Efficiency
     - Scalability

CI Tasks

4. Continuous Inspection
   - Implementation of Design
   - Code Style Standard
   - Code Duplication
   - Code Coverage
   - Bug Prediction
   - Tests Coverage
   - Crypto Implementation Correctness

CI Tasks

5. Continuous Delivery
   A. Continuous Deployment
     - Build labeling
     - Multiple environments
     - Hierarchical deployment
     - Deployment roll back

CI Tasks

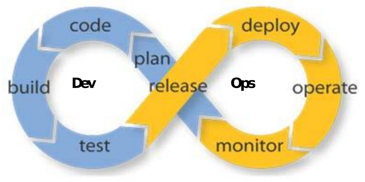
5. Continuous Delivery
   B. Continuous Release
     - Semantic versioning management (e.g. major, minor, revision, build#)
     - Packaging
     - Supported environments
     - Releases availability
     - Releases recall
     - Patches management

CI Tasks

6. Continuous Feedback
   - What feedback is sent?
   - To whom the feedback is sent?
   - When should feedback be sent?
   - How feedback is sent?
   - What kinds of responses are expected?
   - How fast the responses are expected?

DevOps

- "A culture, movement or practice that emphasizes the collaboration and communication of both software developers and other IT professionals while automating the process of software delivery and infrastructure changes" [Schmidt 2016]



Appendix K ICAgile ICP-FDO Learning Objectives (Dec 2017)

1. Version Control
   - 1.1. Commit Everything
   - 1.2. Infrastructure as Code
   - 1.3. Working on Main line
2. Mindset and Principles
   - 2.1. DevOps Principles
   - 2.2. Systems Thinking
   - 2.3. Definition of Done
   - 2.4. Communication
   - 2.5. Collaboration
   - 2.6. Reduced Risk
   - 2.7. Small, Frequent Releases
   - 2.8. Feedback Loops
   - 2.9. Continuous Improvement (Kaizen)
   - 2.10. Ingraining the DevOps Mindset
   - 2.11. Measuring DevOps Success
3. Cultural Challenges
   - 3.1. Essential Conflict
   - 3.2. Teams
   - 3.3. Organizational Structure
   - 3.4. Confidence in Automation
   - 3.5. Resistance to Change
   - 3.6. Incremental Implementation
   - 3.7. Time-bound vs. Queue-driven
   - 3.8. Distributed Teams
4. Dependency Management
   - 4.1. Artifact Management
   - 4.2. Third-party components
5. Managing Configuration
   - 5.1. Application Configuration
   - 5.2. Environment Management
   - 5.3. Feature Toggles
   - 5.4. Configuration Management Tools
6. Practices of Continuous Integration
   - 6.1. Principles of Continuous Integration
   - 6.2. Commit Code Frequently
   - 6.3. Prioritize Fixing the Build
   - 6.4. Writing Automated Developer Tests
   - 6.5. All Tests and Inspections Much Pass
   - 6.6. Run Private Builds
   - 6.7. Avoid Getting Broken Code
7. Build Automation
   - 7.1. Build Tools
   - 7.2. Minimize External Requirements
8. Quality Assurance
   - 8.1. Static Analysis
   - 8.2. Development Standards
   - 8.3. Continuous Feedback
   - 8.4. Distributed Teams
9. Philosophy and Mindset
   - 9.1. Definition of Continuous Delivery
   - 9.2. Continuous Delivery vs. Continuous Deployment
10. Principles of Continuous Delivery