MODERN COMPUTING PARADIGMS:
MOBILE APPLICATIONS AND INFRASTRUCTURE AS CODE

A Thesis

Submitted to the Graduate School
of the University of Notre Dame
in Partial Fulfillment of the Requirements
for the Degree of

Master of Science

in

Computer Science and Engineering

by

Aaron Huus

______________________________
Gregory Madey, Director

Graduate Program in Computer Science and Engineering
Notre Dame, Indiana
April 2013
MODERN COMPUTING PARADIGMS:
MOBILE APPLICATIONS AND INFRASTRUCTURE AS CODE

Abstract
by
Aaron Huus

The computing world is constantly changing and requires the use of new technologies to solve emerging world problems. This thesis specifically looks at how to apply two new technologies, mobile computing and infrastructure as code, to address the issue of crumbling structures in Haiti and the provisioning of virtual machines for collaborative team work.

An Android mobile application was created and will be tested during the summer of 2013 by a small group of engineering students in Ecuador, Uganda, and Costa Rica. The application will eventually be used in Haiti to guide a Haitian citizen through an assessment to obtain data that civil engineering experts will later use to construct a structural model and identify areas at risk of collapse.

DevOps is a general movement from a traditional development approach to an agile approach. Within this movement, exists a methodology known as infrastructure as code that focuses on how the infrastructure is created, automatically provisioned and maintained. A provisioning language, Puppet, is used to provision a virtual or physical machine with specified configuration settings. A benefit of this approach is the developers, testers, and system administrators all work on identical computing environments.
DEDICATION

I wholeheartedly dedicate this thesis to my loving fiance / soon to be wife Emily, my parents Paul and Melissa, and my brother Matthew. Without their unending love and support throughout my education and writing, none of this would have been possible.
CONTENTS

FIGURES . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . ix
TABLES . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . x
ACKNOWLEDGMENTS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . xi

CHAPTER 1: INTRODUCTION . . . . . . . . . . . . . . . . . . . . . . . . . 1
  1.1 Computing Paradigms . . . . . . . . . . . . . . . . . . . . . . . . . 1
  1.2 Haiti Infrastructure Surveyor Mobile Application . . . . . . . . . . 1
    1.2.1 Purpose . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
    1.2.2 Importance . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
    1.2.3 Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
  1.3 Infrastructure as Code . . . . . . . . . . . . . . . . . . . . . . . . . 3
    1.3.1 Purpose . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
    1.3.2 Importance . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
    1.3.3 Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
  1.4 Thesis Overview . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

CHAPTER 2: RELATED WORK . . . . . . . . . . . . . . . . . . . . . . . . . 5
  2.1 Mobile Computing . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
    2.1.1 Crowdsourcing . . . . . . . . . . . . . . . . . . . . . . . . . . 5
    2.1.2 Data Obtainment Via Mobile Applications . . . . . . . . . . . 6
  2.2 Infrastructure As Code . . . . . . . . . . . . . . . . . . . . . . . . . 8
    2.2.1 Machine Provisioning . . . . . . . . . . . . . . . . . . . . . . 9
    2.2.2 System Administration . . . . . . . . . . . . . . . . . . . . . . 10

CHAPTER 3: HAITI INFRASTRUCTURE SURVEYOR MOBILE APPLI-
ICATION . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
  3.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
    3.1.1 Chapter Overview . . . . . . . . . . . . . . . . . . . . . . . . . 15
  3.2 Background . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
    3.2.1 Building Codes . . . . . . . . . . . . . . . . . . . . . . . . . . 15
      3.2.1.1 United States . . . . . . . . . . . . . . . . . . . . . . 16
      3.2.1.2 Haiti . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
    3.2.2 Structural Analysis Problem . . . . . . . . . . . . . . . . . . . 18
CHAPTER 4: INFRASTRUCTURE AS CODE: A SURVEY FROM INCEPTION TO IMPLEMENTATION

4.1 Introduction

4.1.1 Chapter Overview

4.2 Background

4.2.1 System Administration

4.2.2 DevOps

4.2.3 Infrastructure as Code

4.3 The Need For Infrastructure as Code

4.3.1 Users

4.3.1.1 Programmers

4.3.1.2 Quality Assurance Personnel

4.3.1.3 Deployment Personnel

4.3.1.4 System Administrators

4.3.2 Advantages

4.3.3 Disadvantages

4.3.4 Alternatives

4.4 Infrastructure as Code Implementation Requirements

4.4.1 Operating System

4.4.1.1 Virtual Machine

4.4.1.2 Physical Machine

4.4.2 Software Configuration Management
4.4.2.1 Puppet ............................................. 48
4.4.2.2 Chef .............................................. 48
4.4.2.3 CFEngine ......................................... 48
4.4.3 Version Control .................................... 48
4.5 Implementation Use Case ................................ 49
4.5.1 Design Decisions .................................... 49
4.5.1.1 Software Configuration Management Tools .... 49
4.5.1.2 Virtual Machine Usage .......................... 50
4.5.2 Configuration ....................................... 50
4.5.2.1 Vagrant ........................................... 50
4.5.2.2 Puppet Client ..................................... 51
4.5.2.3 Puppet Master .................................... 53
4.5.3 Implementation ..................................... 54
4.5.4 Encountered Issues ................................ 57
4.6 Limitations and Future Work ............................ 58
4.7 Summary ............................................... 59

CHAPTER 5: CONCLUSION .................................................. 60
5.1 Haiti Infrastructure Surveyor Mobile Application ...... 60
5.2 Infrastructure as Code ...................................... 60

APPENDIX A: SOFTWARE REQUIREMENTS SPECIFICATION FOR HAITI
INFRASTRUCTURE ANDROID APPLICATION ....................... 62
A.1 Introduction ............................................. 62
A.1.1 Product Purpose ..................................... 62
A.1.2 Product Scope ....................................... 62
A.1.3 Product Prospective .................................. 63
A.1.4 Product Functions .................................... 63
A.2 Interface Requirements ................................... 64
A.2.1 User Interfaces ...................................... 64
A.2.1.1 Parcel Data ........................................ 64
A.2.1.2 Document Observations ......................... 64
A.2.1.3 Photo ............................................. 64
A.2.1.4 Report ............................................. 65
A.2.2 Hardware Interfaces ................................... 65
A.2.3 Software Interfaces ................................... 65
A.2.4 Communication Interfaces ............................. 65
A.3 System Features ......................................... 66
A.3.1 Wizard Walkthrough ................................ 66
A.3.1.1 Description ........................................ 66
A.3.1.2 Stimulus/Response Sequences .................... 66
A.3.1.3 Feature Rationale ................................ 66
A.3.1.4 Functional Requirements ......................... 66
A.3.2 Parcel Data Questionnaire ............................. 67

v
FIGURES

3.1 Map showing the epicenter of the 7.0 magnitude earthquake that devastated Haiti in 2010. 13
3.2 Current homes being constructed in Haiti. 17
3.3 Mobile devices used for testing the HIS application. 20
3.4 Test table created by the EPICS team that was used to display the information and photographs uploaded from the HIS application to the mySQL database. 21
3.5 Structure selection screen capture. 28
3.6 Questionnaire screen capture. 29
3.7 Exterior observations screen capture. 30
3.8 Data submission context menu screen capture. 32
4.1 The DevOps process combines the traditional development and operations teams into one unified team, a DevOps team. 37
4.2 Relationship between development, quality assurance, and technology operations [5]. 38
4.3 Multiple puppet clients may be apart of the same node and multiple nodes are configured to be provisioned from a puppet master. 55
B.1 UML diagram of module tasks, as well as the path through the entire application. 79
B.2 Summary of the exterior observations module where dotted circles represent individual images and boxes represent textual observations. The solid circle represents the central structure being observed which each image and textual observation are correlated with. 80
# TABLES

3.1 MODULE OVERVIEW ........................................... 25
4.1 COMMON VAGRANT COMMANDS ................................. 52
4.2 CREATED PUPPET MODULES ................................. 56
B.1 MODULE OVERVIEW ........................................... 81
B.2 USE RELATIONSHIPS AMONG HAITI APPLICATION MODULES 88
ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Gregory Madey. None of what I have done would have been possible without his unending support and leadership.

I would also like to thank Dr. Tracy Kijewski-Correa, Dr. Alexandros Taflanidis, Dr. Collin McMillan, the EPICS team at Notre Dame consisting of Dolff Hanke, Charles Jhin, Henry Kim, Jonathan Koch, Matthew Maguire, and Michael Powers. Fellow graduate student Ryan Michaels also assisted during the development. Each of these individuals was an instrumental asset during the development of the mobile application. The collaboration with this group of people was crucial while developing the mobile application to ensure the correct design requirements were constantly being implemented. The development process and current state of the application would not be the same without these crucial individuals.

Finally, I would like to thank the Center for Research Computing (CRC) at the University of Notre Dame, specifically Dr. Timothy Wright, Dr. Charles Vardeman, and Alexander Vyushkov. This group was a tremendous help in brainstorming and always available for collaboration while determining the specific requirements desired by the CRC. Without this group, the completion and beginning stages of implementation of the new infrastructure as code policy would not have been possible.
CHAPTER 1

INTRODUCTION

1.1 Computing Paradigms

Research within computer science is extensively broad and promotes collaboration with a variety of disciplines. The collaboration between disciplines is precisely what makes research in this field relevant to a large number of people. Additionally, incorporating a client or end user into the research and development aspect provides an alternative perspective to the problem and helps define the functional requirements of the solution. Two areas that are becoming increasingly more prevalent as our society transitions to technology dependence are mobile application development and development operations (DevOps) or more specifically infrastructure as code.

1.2 Haiti Infrastructure Surveyor Mobile Application

The Haiti Infrastructure Surveyor (HIS) mobile application was created through the collaboration with Dr. Tracy Kijewski-Correa, Dr. Alexandros Taflanidis, the Notre Dame Engineering Projects in Community Service (EPICS) team, and fellow graduate student Ryan Michaels. The development of the HIS application was the result of a distinct purpose, addresses an important issue, and produces measurable results that may improve the safety of structures within Haiti.
1.2.1 Purpose

The purpose of developing the HIS mobile application is to use untrained Haiti community members to gather structural data about various buildings within Haiti. Structural data collected by these community members is uploaded to a backend mySQL database which civil engineering researchers will have access to. Through the use of crowd sourcing, the civil engineering analysts are able to obtain more structural data than if they had to train and deploy their own data collection personnel.

1.2.2 Importance

With the data collected, the civil engineers are able to perform a structural analysis on the building to determine the likelihood of collapse. Early identification of these high-risk buildings allows for preventative counter measures that improve the structural stability of the buildings to be advised. The safety of the Haitian community members is directly impacted through the process to improve the stability of their homes and avoid preventable collapse or damage.

1.2.3 Results

The preliminary development of the application has been completed and now the graphical user interface is being updated to help improve the ease of use and overall user interaction experience. After the application and backend mySQL database have been tested, it will be deployed with a small group of researchers to Ecuador, Uganada, and Costa Rica during the Summer of 2013 for a more in depth use case study.
1.3 Infrastructure as Code

DevOps is a process that helps facilitate the coordination between the development and operations teams. Infrastructure as code is one of many tools that the encompassing field of DevOps uses to help maintain a consistent development, testing, and production environments throughout the life cycle of a software project. The Center for Research Computing (CRC) at the University of Notre Dame is involved with many development projects and will benefit greatly from the implementation of this new approach.

1.3.1 Purpose

The purpose of infrastructure as code is to create a constant computing environment for developers, testers, and system administrators. In doing so, this helps to increase the collaboration between these teams by ensuring that any development or changes committed by one of the groups of people will be compatible with every group. Also, infrastructure as code improves the recovery time after a system crash through the automatic provisioning of machines.

1.3.2 Importance

Due to the fact that the CRC organization is a center located on a college campus, they have to manage the potential turnover of their employees. One reason the CRC chose to implement infrastructure as code was to improve the process by which new employees are trained onto a project. With infrastructure as code, a new developer or tester does not need to spend time learning how to configure their machine to the team’s specifications because it is automatically done through machine provisioning. Additionally, a source code repository of infrastructure modules is created that allows new development teams to build their infrastructure from the ground up in a modular fashion.
1.3.3 Results

The CRC is continuing to work on setting up the basic infrastructure as code methodologies and implement this approach with new and existing software development projects. A presentation was given to the developers who will be utilizing the new approach which discussed the benefits and technical details of infrastructure as code as well as providing links to further resources. Along with the presentation, a short demonstration was shown that reiterated the usefulness of adopting the new methods into their daily routine.

1.4 Thesis Overview

Chapter 2 performs a literature survey with regards to both mobile computing as well as infrastructure as code. Chapter 3 further discusses the motivation behind the development of the application along with specific implementation and deployment details. Chapter 4 discusses the relationship between the concept of DevOps and infrastructure as code along with an implementation example. Finally, Chapter 5 reiterates the purpose for researching these paradigms and performing the case studies, as well as summarizes the currently completed work.
CHAPTER 2
RELATED WORK

2.1 Mobile Computing

The mobile computing paradigm has been used to solve a variety of different problems utilizing numerous approaches. The two main approaches this thesis focuses on are (1) crowdsourcing and (2) data obtainment via mobile applications.

2.1.1 Crowdsourcing

Crowdsourcing, more specifically citizen engineering for our application, is a method that leverages a large number of individuals to help solve a problem. In today’s society where people are constantly distracted by their mobile phones or other internet enabled devices, people appear to have extra time that is used at their leisure. It is precisely this idea of people having a cognitive surplus [98] that allows for the crowdsourcing approach to be successful.

James Surowieck, in his book *The Wisdom of Crowds* [104], demonstrates that under the right circumstances groups are remarkably intelligent, and often even smarter than the smartest people in them. Harnessing the collective intelligence of a group is important, because no one individual knows everything and each member always has something to contribute [83]. Capitalizing on the power of collective intelligence has resulted in numerous successful and profitable examples throughout a variety of industries such as Threadless [14], and InnoCentive [10].

The benefits of crowdsourcing are also observed in products with a strong research or educational component such as Stardust@Home [110], CamClickr [108],
and eBird [102]. Each of these projects utilizes common citizens to help collect, manage, observe, and store various aspects of data to accomplish a common research or educational goal. Similarly, through citizens participating in the crowdsourcing process, advances have been made in civil infrastructure monitoring [55, 88], political debate performance [54, 94], and emulating a database in a crowd [60, 86]. Many problems can be broken down into simple enough steps where they may be outsourced to the crowd and capitalize on the productivity of the parallelization of a task.

A problem with the crowdsourcing approach to arrive at a solution is the inability to perform quality control throughout the design process. This is an issue that results from the variety of users that will participate. Some of these users will be experts in the area, some will be hobbyist, and some may even have a malicious intent. Chevrolet tried to crowdsource the creation of one of their 30 second commercials for the 2007 Chevy Tahoe, but the participants instead created 30 second clips that ranted about issues ranging from the current political policy to the state of the overall automotive industry [73]. The potential for subquality work must be expected when crowdsourcing a solution from a large number of people, but by establishing a realistic workflow that maintains user enthusiasm helps improve the overall quality received.

2.1.2 Data Obtainment Via Mobile Applications

The use of mobile devices to remotely obtain data is becoming increasingly prevalent. These mobile devices have been used to assist in data obtainment across a variety of fields including public health and crisis response. One purpose of every application used for data obtainment is to create a centralized data repository that people may observe or analyze in order to identify trends or infer a solution to the problem.

Public health is “the science and art of preventing disease, prolonging life, and promoting health through the organized efforts and informed choices of society, orga-
nizations, public and private, communities, and individuals“ [111]. Public health has been able to improve the outreach into the community and gather more information as mobile applications have become more prevalent. The Ushahidi Haiti Project was used to produce a crisis map directly after the Haiti earthquake [90]. The crisis map was used to help improve situational awareness from event data submitted by the application users. Similarly, maps of massively collected information are also used to help detect disease outbreaks [61].

Community health workers (CHWs) are individuals who are recruited by a primary care facility in developing nations to provide primary care for the surrounding communities [103]. Two developed applications, FrontlineSMS [85] and GeoChat [75], have greatly improved not only the communication between the CHWs and the primary care facility, but also the overall quality of care provided to the community. FrontlineSMS is free open source software that is used to distribute and collect information via text messages. GeoChat is an open source group communication tool that helps team members communicate and keep track of what everyone else is working on. An added benefit of using the GeoChat service is that it is cross platform compatible so whether a computer, mobile phone, or other internet enabled device is used the user is able to efficiently communicate with the entire team. Through the utilization of these applications it has been possible to target specific communities that are experiencing a disease outbreak, as well as observe outbreak trends and mitigate its effect.

In addition to the community health application, both FronlineSMS and Geochat have been applied to improve the coordination and communication within a crisis response team. At the time of the Haiti earthquake, approximately 85 percent of Haitian households had access to a mobile phone with Digicel being the main cell phone service provider. A system was set up that allowed Haitians to submit crisis alerts for free via text message as a result of the coordinated efforts of a team at
Fletcher School of Law within Tufts University, the U.S. State Department, Digicel and the FrontlineSMS software [68]. Similar systems have been established during a crisis such as COMBINED [105], ArcPad [89], GeoMIP [84], and WORKPAD [50]. Mobile applications and stand-alone services have also been established that lead citizens effected by the disaster to safety and harness the information they observe about the situation [65, 66]. Each system was applied under different circumstances, but each application had the purpose of ensuring that the crisis was handled in an effective way [115]. In doing so, this improves the communication between emergency team members as well as the overall efficiency of the crisis response system.

Mobile applications have also been used to facilitate the crowdsourcing approach to the collection of photographs. A web-based company, iStockphoto [29], was launched in February 2000 with the goal of collecting a large amount of user submitted photographs. The photographs are screened for quality before being accepted into the database and the photographer receives 20 percent of the purchase price for any of their photographs that are downloaded. Another popular application which aggregates user submitted photographs into a central location is Flickr [28, 71]. Over 6 billion images have been uploaded through this service with 190 million of these having some type creative commons license. LocoBlog [41] is a mobile blog application that allows user to submit their photos to a personal blog directly from their mobile device. The interface allows photos, textual information, and the GPS coordinates of the picture to be stored and accessed later.

2.2 Infrastructure As Code

Infrastructure as code is the process which helps ensure the collaboration between development, quality assurance, and operations teams is conducted using identical computing environments. The main aspect of this approach that this thesis focuses on are (1) machine provisioning and (2) system administration.
2.2.1 Machine Provisioning

Machine provisioning is the process by which virtual or physical machines are automatically provisioned and configured through the use of a system configuration language such as Puppet [11], Chef [4] or CFEngine [3]. Each of these languages implements the client-server architecture in which a central server is in charge of scheduling the provisioning of each of its client machines [79].

Networks that consist of multiple clients and are used for a variety of applications benefit from the recently published addons Control-tier [24] and Capistrano [62]. These two are additions to Puppet and Chef respectively and help coordinate and scale the administrative tasks across multiple machines. SmartFrog [64] is an entirely distributed architecture that is used to automate the provisioning and configuration of multiple machines.

The machine provisioning methodology is widely used [40] to provision virtual machines (VMs) used to host cloud services such as CloudSim [49], OPTIMIS [57], and FlexPRICE [69]. When provisioning cloud services, other attributes in addition to configuration management of the system are taken into consideration. These considerations include power optimization [78], resource optimization [99], and thermal management [92]. Monitoring the status of all of these attributes help reduce the environmental impact of the servers [91] while also minimizing operation costs.

The administration of cloud services may also be offloaded from a hosting company such Amazon EC2 [2] or Rackspace [12] to an independent user. VMPlant [80] handles VM creation and hosting for traditional operating system platforms. Using the VMPlant architecture, users are able to define customized workspace environments that may be shared with others for reuse. An example of an application that utilizes VM provisioning and cloud service technology is In-VIGO [38]. The application dynamically instantiates a web accessible virtual workspace that allows users to graphically access virtual applications and remote filesystems.
2.2.2 System Administration

The field of system administration has also benefited from the utilization of machine provisioning. With the increase in complexity of computing systems, new approaches [42] are looking at automating the system administration process [6, 13, 72]. These software solutions help automate the once manual tasks of designing, configuring, troubleshooting, and maintaining computer systems. The simplistic approach to machine configuration and software package installation process is to generate shell scripts or perform a file distribution from a central repository [39, 58, 59, 96].

The system administration process focuses on the stability of entire systems by monitoring all changes to improve the overall reliability of the system. A test-driven or diagnostic approach is beginning to be implemented [47] through the use of tools such a CFEngine [3] and Puppet [11]. A major concept in the test-driven approach deals with the idea of maintenance and the continual provisioning of the system. The automation of machine maintenance transforms the issue of when to perform maintenance to being able to predict when repairs are required [82, 114]. Utilizing this “predict and prevent” e-maintenance methodology will help improve the overall asset utilization within a facility.

A similar approach to the system administration process is performed by semi-automating the monitoring of the health of the computing system. Swatch [100] helps the system administrator by sending out e-mails or alerts when the health of the overall system begins to suffer. Another tool, PARMON [48], allows for the effective monitoring and control of a large cluster of machines. One of the new features offered is that PARMON supports monitoring at three different levels: entire system, node and component level. Automated scanning of event logs is also used to detect what is wrong with a system [107, 112, 113]. The benefit of these scanning algorithms is that system logs today can become very large and the automated process of extracting important information captures information that may have been missed otherwise.
The overall purpose of these tools is to semi-automate the process of monitoring the health of a system and only prompt the system administrator needs to verify or act on a system health issue.

Monitoring the health of a system is useful, but the ideal approach to system administration would be to build a machine that is able to configure, heal, optimize, and protect itself. This approach is defined as autonomic computing [76] which is a relatively new paradigm that reduces software maintenance and management costs [81]. Autonomic computing is being applied throughout a variety of applications with the main focus to self-configure [51, 87], self-heal [63, 93], self-optimize [109], and self-protect [52]. Maintaining systems in such a way will reduce the time system administrators spend on both configuration and maintenance tasks.
CHAPTER 3

HAITI INFRASTRUCTURE SURVEYOR MOBILE APPLICATION

3.1 Introduction

On January 12, 2010, an estimated 220,000 people died within 30 seconds during the 7.0 magnitude earthquake that devastated the nation of Haiti [7]. The earthquake’s epicenter, as shown in Figure 3.1, was determined to be the seaside town of Leogane which is located 15 kilometres southwest of Port-au-Prince, Haiti’s capital. Leogane had a population of 134,000 and was the worst affected area with 80 to 90 percent of buildings suffering damage. These staggering structural damage percentages were observed throughout many cities including Carrefour, which had a population of 334,000 individuals, and observed 40 to 50 percent of structures destroyed [8]. Poor structural stability was a major cause of the observed damage.

As a result of all of the reported damage, it has been estimated that 1.5 million people instantly became homeless and 3.5 million people were affected [7]. With Haiti in despair and nearly 40 percent of the entire country drastically affected by the earthquake, the United States, the United Nations, and many other countries and organizations rushed to provide humanitarian aid and assist in any way possible.

The first country to provide aid for Haiti was the neighboring country of the Dominican Republic by supplying food, medicine and rescue crews [9]. Many other countries rushed to the aid of Haiti some of which included Iceland, China, and the United States. Iceland was there within 24 hours of the earthquake [31], China sent a 50 member international rescue team [21], and the United States American Red
Figure 3.1. Map showing the epicenter of the 7.0 magnitude earthquake that devastated Haiti in 2010.

Cross ran out of medical supplies and was accepting public donations after one day of provided aid [35]. The unified global effort taken to provide aid to the nation of Haiti was truly incredible, however one must begin to understand why this earthquake was so devastating to begin with.

There are two main reason’s why the Haiti earthquake in 2010 was so devastating. The first reason is that the epicenter of the earthquake was located near such a densely populated area and occurred roughly 10 kilometres below ground. A 6.8 magnitude earthquake hit Seattle, WA in 2001 and was much less severe, because it happened approximately 60 kilometres below ground [37]. The increased depth allowed a lot
of the earthquake’s energy to dissipate before reaching the surface and resulted in significantly less devastation.

Another reason why the Seattle earthquake was less severe was because Seattle’s buildings are constructed to withstand the shaking and twisting accompanied with an earthquake [37]. The poor structural infrastructure in Haiti is the major reason why the earthquake was so devastating. Haiti is the poorest country in the western hemisphere [23] with 80 percent of the country living under the poverty line [22]. Countries that live within these poverty levels do not have the necessary educated personnel or resources to build structures to the appropriate specifications to withstand high magnitude earthquakes. The poor construction of Haiti’s infrastructure is the main problem that needs to be solved in order to prevent a disaster of similar magnitude from reoccurring in the future.

One approach to address the issue of poor construction within Haiti is to implement a crowdsourcing approach to gather structural data of structures in Haiti. There are currently no enforced building codes within the country, therefore in order to run a structural analysis of the building, the structural information needs to be observed and documented by a surveyor. Conversely, when a structural analysis is performed in the United States certain assumptions may be made about the structure. These assumptions are based on knowing the structure has been built following the standardized building codes and therefore due to these assumptions that may be made far less measurements are needed for a structural analysis in a nation with established building codes.

The developed mobile solution is to involve local community members by providing them with an android application to guide them through the data acquisition process for at-risk of collapse structures within their community. The data may then be analyzed in order to direct the structure owner to local resources or remedies to address the structural inefficiencies found. The end goal of this application is to facili-
tate the data obtainment stage in order to help improve the overall structure integrity of Haitian buildings and reduce the likelihood of the catastrophic consequences of a natural disaster from occurring again.

3.1.1 Chapter Overview

The remainder of this chapter is organized by first providing a comparison between the United States and Haiti government as well as their respective infrastructure standards. The mobile solution is then described followed by the detailed software development methodology followed during the development of the mobile application. The mobile application functionality and graphical user interface are then described in detail and the chapter concludes with possible directions for future research and an overall progress summary.

3.2 Background

The earthquake damage that devastated Haiti in 2010 was undeniably catastrophic and resulted from the poorly constructed infrastructure of Haiti’s buildings as well as the lack of enforced building codes. Building codes vary greatly between the United States and Haiti. The inability to enforce building codes in Haiti also effects the process of how a structural analysis may be performed on buildings.

3.2.1 Building Codes

Countries around the world have different ways of mandating how structures are constructed within their country. The United States [30] building codes are very detailed and well documented. Haiti’s lack of building codes is due to the lack of government support to enforce them.
3.2.1.1 United States

The United States uses model building codes developed by the International Code Council (ICC). The ICC’s mission is to provide the highest quality codes, standards, products and services for all concerned with the safety and performance of the built environment [30]. In doing so, the ICC has a comprehensive set of publications some of which include:

- International Building Code (IBC)
- International Residential Code (IRC)
- International Fire Code (IFC)
- The International Energy Conservation Code (IECC)

The IBC is in use or adopted in 50 of the United States, the IRC is in use or adopted in 49 of the United States, the IFC is in use or adopted in 43 of the United States, the IECC is in use or adopted in 46 of the United States and all of the codes are in use or adopted within the District of Columbia.

3.2.1.2 Haiti

There are currently no national building codes used within Haiti. If technical standards are used, the decision to which standards to use are determined by the engineers responsible for constructing the structure [25]. University of Arizona researchers are working on creating building codes tailored for Haiti that are low-cost and utilize the skill set and resources of the local community. Simply implementing harsher building codes will not solve the problem, because there is not enough education, money, and physical resources to abide by the stricter building codes [19]. The main goal the Engineering to Empower (E2E) group at the University of Notre
Dame [26, 77] and of the University of Arizona research group is to allow the construction of structures in Haiti that will not fail during an earthquake as well as to make the structure something a Haitian family would enjoy living in. Figure 3.2 shows the houses that are currently being constructed in Haiti. The problem is that these structures are being constructed in the same way that they were before the earthquake occurred which is very endangering to the families that will live inside these structures.

Haiti also faces the issue that their government does not have the needed stability to enforce any mandated building codes. Without a strong central government, individual surveyors are forced to manually collect data about the structure under investigation for stability in order for a structural analysis to be performed.

Figure 3.2. Current homes being constructed in Haiti
3.2.2 Structural Analysis Problem

A structural analysis is a method civil engineers use to assess the structural stability of a structure under different stresses and strains. The approach first requires a computer-based structural model of the building to be created. The task of obtaining the necessary building specifications and input requirements for the model creation varies between the United States and Haiti.

In the United States, the data obtainment is very straightforward and certain assumptions may be made due to established building codes. In Haiti however, these building codes are not enforced so measurements and observations need to be obtained manually from an individual surveyor. This poses a problem because for each structure that needs to be analyzed, an educated surveyor will need to be involved and there are a limited number of these individuals.

3.3 Mobile Solution

A major problem that exists during the collection of structural data is the need for educated personnel that know what data to collect. A solution to this issue would allow the Haiti community members to obtain the desired data by following an intuitive mobile application. The data is submitted to a backend database which the civil engineers may access in order to perform a structural analysis on the structure. The goal of this analysis is to determine if the building is structurally safe and to direct the surveyor to local resources and solutions to help improve the structural stability. The current version of the application was developed to provide the initial functionality desired in the application such as capturing input from the surveyor in multiple formats and submitting this information to a server. In order for a full structural analysis to be performed, further development of the application will need to take place and allow the application to prompt the surveyor for more specific exte-
rior and interior questions. These additional questions will be identified as important by the civil engineering research team analyzing the data and may include questions regarding wall dimensions, type and thickness of rebar in the walls as seen from the roof, floor plan, and structure building material.

3.3.1 Crowdsourcing

Crowdsourcing, more specifically citizen engineering, for our application refers to the idea of leveraging a large number of ordinary citizens to solve real-world problems [116]. A benefit of this approach for the Haiti structure problem is that it will allow the citizens within the community to help one another and be directly involved with improving the safety of their homes. Also, utilizing the local population for data collection frees up trained research personnel to focus their efforts on performing the structural analysis.

3.3.2 Mobile Application

A mobile application platform was selected due to the required mobility of the developed product. The solution had to be easily portable, document notes, take photos, easy for a user to understand, and have access to the internet. A mobile application perfectly addressed each of these needs. The android operating system was selected because a donation of 100 Motorola XT720s mobile devices compatible smart phones that were made available for the project. The application has also been tested on a Motorola Xoom tablet. These two devices are shown in Figure 3.3.

3.3.3 Backend database

The HIS mobile application requires a location to store all of the information gathered about the structures in Haiti to a centralized location. A backend mySQL database and web interface were created by an EPICS team (an undergraduate team
enrolled in a service learning course) at the University of Notre Dame. The mySQL server is currently hosted on GoDaddy.com and interacts with the mobile application through a PHP file. This is the file that the HIS application interacts with via a HTTP POST action. The PHP file then extracts the needed information from the _POST and _FILES array and stores the values as variables within the PHP script. An initial connection is made to the mySQL database using preconfigured authentication parameters. A SQL query is dynamically created using the variables and executed. Upon completion, the status of the query execution may be verified to see if it has executed successfully. After a successful completion the image files are moved to an image folder located on the server. Finally, to complete the process the connection to the mySQL database is terminated. Figure 3.4 shows what the test table looks like that is used to verify that the pictures taken from the HIS mobile application actually are able to populate a remove mySQL database.

The EPICS team is currently working on improving the web site's interface and has added functionality allowing for the ability to sort and search through the various structures. This work will be completed by May 2013 in order to allow for the application to utilize the new interface before the civil engineering students travel to Equador, Uganda, and Costa Rica.
3.3.4 Structural Analysis

The civil engineering research team will have access to all of the structural data collected by the Haiti community members via the mobile application. With the structure data, an analysis may be performed to determine the risk of collapse. If the structure is determined to be at high-risk, a response is issued to the community member which will direct them to local resources that will help improve the structure’s stability.

3.3.5 Local Resource Allocation

The results from the structural analysis will be analyzed and various recommendations on how to improve the structural integrity of the building will be provided to the local community member in Haiti. The recommendation will consist of contact information for the local resources needed to make the improvements, along with a detailed explanation of nontraditional construction methods to help improve
the safety of the structure. One example of these nontraditional methods has been discussed by Dr. Ahmet Turer, a structural engineer at the Middle East Technical University (METU). Dr. Turer suggests that old recycled tires may be used to reinforce masonry houses [32] by encasing the house walls with the tires which would improve the stability and make them able to withstand seismic shocks.

3.4 Waterfall Software Development Process

The Haiti Infrastructure Surveyor (HIS) application was constructed for Dr. Tracy Kijewski-Correa and Dr. Alexandros Taflanidis in the Department of Civil & Environmental Engineering & Earth Sciences at the University of Notre Dame. The HIS application will be deployed in Haiti to assist with the collection of structural data and has also been the focus of the collaboration with the Notre Dame EPICS team as well as the final project of a software engineering course. For the software engineering course, the class was instructed to develop their software application following the traditional waterfall software development process. Although the waterfall method was used to develop the application up to this point, for continued development it would be advised to switch to an agile software development method approach. This type of approach is more commonly used and allows for incremental design and functionality changes throughout the development of the application.

The waterfall method of the software development life cycle utilizes strict development stages and does not easily accommodate changes in the design requirements. The stages of the waterfall method include requirement specification, design, development, testing, deployment, and maintenance of the application. Not all of these stages were completed within the software engineering course time frame, but the following sections describe how each stage either was completed or will be completed in the near future.
3.4.1 Requirement Specification

The first stage that was performed while designing the HIS application was to clearly define the functional and nonfunctional requirements of the application. Through the collaboration with Dr. Kijewski-Correa and Dr. Taflanidis, the following list summarizes the main requirements that the application must perform and was the focus during the development of the application.

- Simple user interface
- Wizard walkthrough
- Questionnaire
- Document textual notes
- Take a photograph
- Communicate with a backend database

For more details, the requirements document for the HIS application may be referenced in its entirety in Appendix A.

3.4.2 Design

A modular approach was taken during the design of the HIS application by splitting the functionality into five top level modules: structure selection, questionnaire, exterior observations, interior observations, and data submission. Each of these modules communicates with each other in a wizard fashion such that the module elicits a call to the subsequent module to take over directing the users actions.

The structure selection module is responsible for maintaining a list view of previously started, but not submitted, structure surveys and is where the user will always begin interacting with the application. The questionnaire module appears once the
structure selection module identifies which structure the user will be working with. Upon finishing this module, the user is directed to the exterior observation module. It is here where the user begins to take photos of the exterior of the structure and document various observations. The interior module is called on after the completion of the exterior module. This module functions similar to the exterior module in that it requires the user to document the interior of the structure using textual and pictorial observations. After completion of the interior module, the data submission module is called to assist the user in committing the recorded observations to the database. The modules are summarized in table 3.1 below. For more details, section 3.5 describes in detail how each module functions as well as images of the mobile application’s user interface. Also, the design document for the HIS application may be referenced in its entirety in Appendix B.

3.4.3 Development

The HIS application was developed using the android SDK [18], the Eclipse IDE [27], and the Java programming language. HIS will be deployed onto the Motorola XT720 mobile smartphone running Android 2.1 [17] and Motorola Xoom tablet which are displayed in Figure 3.3. Newer mobile devices and android operating systems may still utilize the functionality of the application due to backwards compatibility of the newer versions of the android operating system. The application has also been tested and is functional on a Motorola tablet.

The development process followed the predetermined modules discussed in Section 3.4.2 by encapsulating the application’s functionality into separate files. The purpose of functionality separation into distinct files was to simplify the testing and maintenance of the application. Also, the separation of functionality into distinct files helped improve the readability of the code and therefore make the transition
TABLE 3.1

MODULE OVERVIEW

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structure Selection</td>
<td>Allows the user to add or select a structure to begin documenting</td>
</tr>
<tr>
<td>Questionnaire</td>
<td>Gathers basic information about the current structure being observed</td>
</tr>
<tr>
<td>Exterior Observations</td>
<td>Directs the user through the process of documenting the exterior</td>
</tr>
<tr>
<td>Interior Observations</td>
<td>Directs the user through the process of documenting the interior</td>
</tr>
<tr>
<td>Data Submission</td>
<td>Collects the acquired data about the current structure and submits it to a backend database</td>
</tr>
</tbody>
</table>

of a developer onto the project more efficient. To assist with team collaboration, a project within Google code was created and the SVN repository was used to host the HIS application source code. The SVN repository allowed the developer to choose which operating system they preferred to develop on including the Mac OS X, Linux, or Windows operating systems.

For more details, the source code for the HIS application may be referenced in its entirety in Appendix D. Additionally, Appendix C provides a guide to getting started with android mobile application development.

3.4.4 Testing

The specific application requirements were tested individually by manually checking to see if all of the functionality desired by Dr. Tracy Kijewski-Correa and Dr. Alexandros Taflanidis had been implemented and functioned as desired. In the near
future, a trial release of the application to a small set of researchers traveling to Ecuador, Uganda, and Costa Rica. The final stage of testing will consist of meeting with Dr. Tracy Kijewski-Correa and Dr. Alexandros Taflanidis and observe them use the application to obtain any final feedback and identify if any features did not function properly.

3.4.5 Deployment

The HIS application will be installed onto approximately 100 Motorola XT720 smartphones and distributed to various community members in Haiti. The community member selection process is based on how well the selected community member is involved with the betterment of their community and the ability for them to effectively utilize the HIS mobile application. An incentive such as free cell phone service will be provided to the individual to help ensure the quality of work being completed.

3.4.6 Maintenance

The specific details of the maintenance of the HIS application still need to be determined, however there are many approaches that are suitable. The main issue of maintenance deals with how new versions of the software are going to be pushed to each of the mobile phones. One way of dealing with this is to publish the application in the Google market. In doing so, however, would make the application public which is not the current focus for development as it will be used as a private research tool.

Another solution is to host the update file on a privately owned web server and have the HIS application check periodically if there is an update available. When an update is available the user will be prompted as to whether or not they would like to install the update. The benefits of this approach are the application update file will remain on a private server and the owners may specify who has access to downloading the update. A private server is very useful because the application should only be
distributed to the the 100 mobile phones originally provided to the Haiti community members.

3.5 Mobile Application

The HIS mobile application directs a Haitian community member to perform a structural survey on a building that appears to be at risk of collapse. The application’s main purpose is to allow the user to maintain multiple structure surveys. Within each of these surveys the HIS mobile application directs the user to answer a questionnaire, obtain exterior photos and measurements, obtain interior photos and measurements, and submit all of this data to the remote backend database.

3.5.1 Structure Selection

The structure selection module is the first interface produced by the application. The purpose of this module is to allow the user to add a new structure or select which previous structure they are working on if they are surveying multiple structures simultaneously. The structure list will also show previously submitted structures for a designated amount of time before they will be automatically removed from the list. An example of this interface is shown in Figure 3.5.

A long click on one of the structures in the list will bring up a context menu prompting the user with the options to either upload, open, edit, or delete. The upload functionality is discussed later in Section 3.5.5. The open option does the same as single clicking on the original list item. The edit option allows the user to edit the basic name and description of the structure list item and the delete option deletes the structure from the list. The list also displays the progress of the survey for each structure in the upper right corner. The possible values for the status of a structure are New, In Progress, Complete, and Submitted.
3.5.2 Questionnaire

Once a structure has been selected, the user is prompted with three screens of questions. The purpose of the questionnaire is to gather the information about the structure that cannot be obtained through an observational note or photograph. Figure 3.6 displays an example of the first screen of the questionnaire.

Not all of the questions are required, but the application does prevent you from proceeding if a required question is not answered. Also, the correct input keyboard is brought up based on whether a numerical or textual answer is expected. One of the most important design decisions when creating the questionnaire module was to abstract the questions into a separate file and avoid hard coding them into the application. The reason for the abstraction is that it will make changing the language
of the application to Creole as simple as changing one file.

![Surveyor Form](image)

Figure 3.6. Questionnaire screen capture

### 3.5.3 Exterior Observations

The exterior observations module is responsible for directing the surveyor to take pictures of the exterior of the structure being observed. The first screen the user will see is a picture of a house surrounded by four buttons each marked with a red X. The red X indicates that a picture has not been taken for that side of the structure. The user then selects one of the buttons and the camera is initiated. Once the picture
has been taken and the user confirms the picture quality, the red X will turn to a green check mark. An example of this is shown in Figure 3.7. The purpose of the red X and green check mark is to visually show the user where they are in the exterior photo obtainment process.

Figure 3.7. Exterior observations screen capture
3.5.4 Interior Observation

The current version of the HIS application does not implement an interior observation module. The next step in the development process is to begin creating this module. The interior observation section of the application will prompt and clearly direct the user to take pictures, document observational notes, and acquire measurements of the interior walls and structural composition. The interface will have a similar theme to the currently developed modules specifically implementing the dynamic image buttons which help indicate the current status of the structural survey.

3.5.5 Data Submission

The data submission functionality is the most crucial aspect of the entire application. Without the ability for the application to upload the collected information from the Haiti surveyor, then all of the photos, observational notes, and measurements collected are useless. The context menu is used to start the upload action and is activated by a long press initiated by the user on the structure from the structure list in which they wish to submit the data. An example of the context menu interface is shown in Figure 3.8.

3.6 Future Work

The Haiti infrastructure surveyor application may still be expanded to include additional features and allow the surveyor to acquire more data about the structure. The more information and documentation gathered about the structures that are being observed, the better the structural analysis performed will be. The improved results from the analysis will improve the ability to provide useful information about local resources to the household owners in Haiti as well as improve the overall safety.
of the community.

The actual deployment of the application onto the mobile phones and distribution into Haiti will also need to be performed in the near future once the development stage of the application is complete. A test deployment may be helpful where a smaller subset of the phones are distributed to community individuals and their usage feedback is taken into consideration. From this feedback, an updated version of the application should be created in order to work out any design or functionality issues observed. One issue that cannot be tested before deployment is if the connection speed achieved during the uploading of structural data via WiFi will be fast enough for general use while in Haiti.
3.7 Summary

Haiti is still recovering and observing the effects from the earthquake that devastated their region in 2010. The problem is that without a strong central government to enforce a set of building codes, structures are being built or repaired through the same methods used before the earthquake. These construction methods result in structures that are structurally unsafe and will collapse again if another earthquake occurs. The proposed mobile solution addresses the issue of poor infrastructure by obtaining the structural data and determining which local resources or construction methods may be implemented to help improve the safety of the community. Governmental change and stability takes time. Until the nation has a way of enforcing safer building codes on their own, the mobile application directly provides a service that will measurably improve the safety of the community and prevent the level of devastation observed from the 2010 earthquake from occurring again.
CHAPTER 4

INFRASTRUCTURE AS CODE: A SURVEY FROM INCEPTION TO IMPLEMENTATION

4.1 Introduction

Currently there is a growing movement for developers, testers, quality assurance, and system administrators to all work together throughout the development of software through a movement known as DevOps [74]. The term DevOps comes from the idea of improving the interaction between the development and operations teams and is a general movement from the traditional development approach to an agile approach across each team. Within this movement, exists a methodology known as infrastructure as code that focuses on how the infrastructure is created, automatically provisioned and maintained. The traditional approach within a company is to divide the software development process into separate teams by the type of work that needs to be performed. Each team has its own deadlines, requirements, and goals and is assigned a separate manager to motivate and assess the team’s progress towards these milestones. The objective of DevOps is to improve the facilitation and communication between the various teams working together on a software development project.

Issues arise with the traditional approach due to conflicting goals of each team. The development team’s goal is to quickly develop and implement new features where as the operations team goal is to ensure the software is functional and stable. The increase in change to the software puts the operations team at greater risk for not
meeting their expectations. Overall, the issues between development and operations may be summarized as the following two issues [74, p. 6]:

- *Produce change:* Development is pressured to constantly add new features, fix bugs, and update the software based on changing requirements.

- *Minimize change:* Operations would like to minimize changes to the software due to their responsibility of maintaining stability and ensuring the deployed version of the software functions as expected.

An approach called agile software development has been applied to the development process which helps decrease the team separation specifically between the programmers, testers, and quality assurance team. According to the agile manifesto [43], which was written by a group of individuals who proposed many of the agile methodologies, agile software development should focus on four core values:

- Individuals and interactions over processes and tools

- Working software over comprehensive documentation

- Customer collaboration over contract negotiation

- Responding to change over following a plan

Agile development has helped form a more unified development team with the main focus on effectively developing software. However, this approach as it is currently applied, maintains the isolation between the development and operations teams. These two teams still maintain their individualistic project goals and therefore use their own processes and tools which optimize the development process for their team. It is this misalignment of the two team’s goals that adversely effects the overall flow and progress of the project as a whole and encourages people to work against one another as opposed to with one another [74, p. 26].
Issues often originate during the transition of software from the development team to the operations team. A common issue that may occur is that the development team delivers their self-declared fully functioning software to the operations team and the operations team cannot get the software to function properly on the production infrastructure. Another issue that may occur is after deployment of the software, a major bug is found and a patch needs to be applied immediately. Both the development and the operations team would want this patch to be thoroughly tested before publishing the changes to the deployed version. However, it cannot be ensured that the test environment exactly mirrors the production environment, so any tests performed may not necessarily function the same on the production infrastructure.

An attempt to improve the environment replication between development and operations and minimize the number of issues dealing with non-identical test and production infrastructure is achieved through the process known as DevOps. Traditionally, a development team consisted of programmers, testers, and quality assurance personnel while the operations team consisted of system administrators, database administrators, and network technicians. DevOps applies the agile software development practices to both the development and operations teams in an attempt to create one unified team which consists of programmers, testers, quality assurance personnel, and experts from operations all working together to develop software. The overall hierarchy of a DevOps team is summarized in Figure 4.1 and the relationship between development, quality assurance, and operations is depicted in Figure 4.2[5].

The end goal behind DevOps is to streamline the software delivery process and improve the quality of the developed software through establishing common goals, processes, and tools.

The DevOps process unifies the development and operations team’s goals while providing a set of common processes and tools to be used. Infrastructure as code is an important part of DevOps and has the potential to drastically alter the soft-
Figure 4.1. The DevOps process combines the traditional development and operations teams into one unified team, a DevOps team.

ware development process. Infrastructure as code directly addresses the issue of non-identical test and production environments by utilizing software configuration management tools such as Puppet [11], Chef [4], or CFEngine [3] to provision the respective environments to be identical. In addition, infrastructure as code allows for a custom infrastructure to be created and specialized for each individual project.

4.1.1 Chapter Overview

The remainder of this chapter is organized by first discussing the background of various terms and concepts that directly relate to infrastructure as code. An overview of the previous research conducted is discussed followed by the requirements needed to implement the infrastructure as code methodology. Next, the need for the infrastructure as code methodology is addressed specifically focusing on users, advantages, disadvantages, and alternative approaches that may be taken. Finally, the entire process of a case study is described from the design decisions made to common issues encountered throughout the implementation. The chapter concludes with a summary and possible directions that may be pursued for future research.
Figure 4.2. Relationship between development, quality assurance, and technology operations [5].
4.2 Background

The DevOps methodology has effectively merged the two previously independent teams of developers and operations. The advancement of infrastructure as code has also greatly helped current system administrators manage large systems of machines. DevOps heavily relies on automation techniques through continuous integration, test driven development, and the utilization of infrastructure as code across programmers, testers, and system administrators.

4.2.1 System Administration

The field of system administration is defined as a realm of computer science which deals with planning, configuration, and maintenance of computer systems [46]. The history of automated configuring of computer systems [56] dates back to 1987 when a paper presented by Ken Stone [101] described a technique for HP workstation cloning through modifications made by *sed* and updates through the use of *rdist*.

The next major movement in automated configuration of systems and cloning scripts was presented during the Large Installation System Administration (LISA) Conference in 1994. From this conference alone, four configuration systems emerged:

- Anderson’s lcfg [39]
- Harlander’s GeNUAdmin [67]
- Hideyo’s OMNICONFG [70]
- Rouillard and Martin’s Config [96]

Each of these systems approaches the idea of improving the system administration process differently, but at the same time share a very interesting commonality [56]. Each proposed system maintains a central repository of definition and configuration files that describe precisely the roles and how the host machine should be configured.
The intended benefit of having a centralized location of configuration files was to help improve the difficulty as well as efficiency of system wide updates and continual management. Shaddok et al. [97] showed this benefit through the use of their Sasify system which performed a system-wide upgrade of 1500 workstations from a centrally administered location. The benefits of this approach were quickly realized and the idea of maintaining a centrally located configuration repository for host machines became commonplace. Throughout the past years, various software configuration management products such as Puppet [11], Chef [4], and CFEngine [3] have emerged which greatly help provision and maintain systems through the utilization of host machine definitions stored in a central repository.

4.2.2 DevOps

DevOps is defined in Michael Huttermann’s book *DevOps For Developers* [74, p. 4] as a blend of development (representing software developers, including programmers, testers, and quality assurance personnel) and operations (representing the experts who put software into production and manage the production infrastructure, including system administrators, database administrators, and network technicians). The term DevOps was first presented in 2009 by Patrick Debois at a conference in Belgium. Additionally, there were other early adopters of DevOps who helped transform and influence its use into an accepted term [74, p. 5]:

- Patrick Debois ran a session called Agile Operations and Infrastructure: How Infra-gile Are You? at the Agile 2008 conference in Toronto and published a similar paper [53]. He discussed the beginnings of infrastructure as code and how it may be utilized to help improve current agile development and operations.

- Marcel Wegermann published an e-mail list called Agile System Administration
The email list focused on the basic ideas of DevOps as it is applied more towards the operations and system administrative tasks.

- John Allspaw gave a presentation called 10+ Deploys per Day: Dev and Ops Cooperation [1] at the Velocity 2009 conference in San Jose. His presentation highlighted the key features of the new DevOps methodology that would improve overall team efficiency.

- Steven Blank published a book called Four Steps to the Epiphany [44] and Eric Ries published The Lean Startup [95] which each talk about the successful strategies used to effectively deliver a product to the marketplace.

- The 451 Group published the first analyst report on DevOps, The Rise of DevOps [34], in September 2010.

These individuals truly helped shape the definition of DevOps and provided a more concrete definition for this term preventing it from becoming a random buzz word. The DevOps movement is attempting to improve communication between developers and operations to solve issues dealing with fear of change and risky premature deployments [74, p. 8]. A major technical component of DevOps that must be implemented in order to effectively promote collaboration between developers and operations is infrastructure as code.

4.2.3 Infrastructure as Code

In recent years there has been a shift in how system administrators are designing their systems. The idea is to be able to customize the infrastructure to the needs of the software project efficiently and repeatedly [74, p. 136]. Also, infrastructure as code strives to treat the infrastructure in the same way software is developed. This is accomplished by being able to develop the infrastructure from modular pieces with
the final infrastructure represented in source code which may be checked into a source code repository.

There are many possible tools that may be utilized throughout the implementation process of infrastructure as code. However, this chapter will focus on why infrastructure as code is needed as well as demonstrate the use of arguably two of the most useful tools: Vagrant and Puppet.

4.3 The Need For Infrastructure as Code

Infrastructure as code has the potential to greatly improve the efficiency and productivity of a company. However, the initial setup of the tools and services required will take some initial time and money. Therefore, an analysis on who will be using this system, the advantages, the disadvantages, and any alternative approaches must be considered before beginning the implementation process.

4.3.1 Users

The main users of the infrastructure as code method will be members from both the development and operations teams. Each member will utilize the provided functionality differently in order to maximize the effectiveness of their work while still closely collaborating with all team members. Throughout the development and maintenance of a software project the main users which benefit are the programmers, quality assurance personnel, deployment personnel, and system administrators. In some companies it is possible that one person or team will take on a combination of these roles.

4.3.1.1 Programmers

The traditional role of the programmers on a software development project is to write the software as described in the application design requirements and spec-
ifications. In the agile software development approach, the programmer will work conjointly with members from different specialties sometimes including a domain expert or potentially the end user themselves. A programmer utilizes the infrastructure as code methodology to automatically ensure that all of the programmers working on the software are developing under the same development environment. Without this, there is a lot of manual work that needs to be done to ensure and verify that each individual programmer’s development environment has the same packages and software installed as well as maintaining identical versions and configurations across all team members.

4.3.1.2 Quality Assurance Personnel

Once the software is believed to be fully functional, quality assurance verifies that the software functions as intended and is free from any bugs or security vulnerabilities. It is required that the environment that the quality assurance personnel test the software on must be identical to the environment that it was developed on. If this is not the case, then it is possible that certain functionality may be lost or security vulnerabilities unintentionally introduced into the usage of the software. Infrastructure as code undoubtedly verifies that the development and testing environments are identical.

4.3.1.3 Deployment Personnel

At this point in the software development process, the software has been developed, tested and is ready for deployment to the end user. The deployment personnel are responsible for taking the software tested by the quality assurance personnel and placing it on the production infrastructure. Infrastructure as code is helpful because the deployment personnel would not need to waste time consulting developers on the infrastructure they used and perform a manual installation of all the needed
packages and software. Instead, the infrastructure can be checked out from a source code repository and compiled on the production infrastructure similar to the physical source code of the software project.

4.3.1.4 System Administrators

The final stage of the software development cycle deals with maintenance. A system administrator is responsible for ensuring the production infrastructure is functioning appropriately and the system does not produce any security vulnerabilities. If the system crashes, it may be easily restored using the infrastructure as code repository which will ensure that the system is configured to exactly the same specifications as it was before the crash. Also, if the system administrator detects a vulnerability and fixes it on the production infrastructure the change may be propagated to the programmers, quality assurance personnel, and deployment personnel via a commit to the infrastructure as code repository.

4.3.2 Advantages

Observing the needs from the different personnel who utilize infrastructure as code helps reveal its advantages. One major advantage is the drastic decrease of complexity while training a new employee on a project. When all of the infrastructure may be checked out from a source code repository and compiled on the new employee’s machine, it removes the need for the new employee to manually download and configure their work environment themselves.

Arguably the greatest benefit of infrastructure as code is that it ensures that all employees develop, test, deploy, and maintain the software on identical infrastructure disregarding hardware differences. In doing so, this instills a sense of accountability because you know if the software does not work on the test environment then the issue must be fixed by the developers. Similarly, if a bug or vulnerability is found on
the production infrastructure, then the quality assurance personnel must have missed it. Overall, infrastructure as code provides a way for multiple people with differing objectives to work on the same project without having to deal with the issues of maintaining an identical work environment across various teams within a project.

4.3.3 Disadvantages

Infrastructure as code clearly provides a benefit to the company or team upon implementation, however there are some disadvantages to utilizing this methodology. The major disadvantage is that it takes some time and effort to set up the needed technology infrastructure before infrastructure as code is possible. For example, each of the developer’s machines need to be initially set up to interface correctly with the code repository. Also, appropriate training would need to be provided for each employee who would want to use this service to ensure that they utilize it correctly. Finally, documentation materials would need to be created in order to document the new process which would aid in explaining the procedures to new employees in addition to provided training. Although there is a slight overhead at the beginning stages of implementation, organizations may find that the advantages outweigh these disadvantages.

4.3.4 Alternatives

If a company or team does not want to fully commit to implementing the infrastructure as code methodology, there is a non-technical alternative, improved communication. If the development and operations team are both involved throughout all stages of the software design process, it is less likely that there will be an issue with the infrastructure. Although this is an alternative, it is still recommended to implement infrastructure as code in addition to improved communication. The infrastructure as code methodology helps automate the process of maintaining a consistent
infrastructure and frees up time for the team to focus on the more important issues.

4.4 Infrastructure as Code Implementation Requirements

Infrastructure as code has many advantages as previously discussed and should be implemented whenever possible. In order to do so, there are a few essential tools and system configuration tasks that must be completed. The major configuration tasks that need to be performed are the installation and deployment of the operating system, software configuration management, and a version control system. The remainder of this chapter focuses on the utilization of infrastructure as code from the perspective of a programmer.

4.4.1 Operating System

There are a couple of decisions that need to be made before beginning on the installation and deployment of the desired host operating system. First, it needs to be determined which operating system the programmers will develop on and the necessary images and license files must be obtained if necessary. Also, it must be decided whether the development environment will be implemented within a virtual or physical machine.

4.4.1.1 Virtual Machine

The installation of the host operating system (OS) on a virtual machine has many advantages and may be accomplished through a tool named Vagrant [15]. This tool assists in the creation and deployment of virtual machines through the use of configuration files and software configuration management tools. An advantage of using virtual machines for development and testing is it is very easy to reload the virtual machine for any reason. For example, during the development process if something becomes corrupt or if at the beginning of each day the developer wants to
start with a clean development environment the virtual machine may be recreated and all the default settings are restored. Once the tool is installed and configuration and provisioning scripts are in place, loading the virtual machine is as easy as typing one command into the terminal.

4.4.1.2 Physical Machine

The installation of the host operating system (OS) on a physical machine may be accomplished through various approaches with the ideal approach selected depending on how many OS installations will need to be executed. If the number of OS installations is relatively small the best approach is to manually install the OS on each of the developer’s machines [45, p. 124]. However, if this becomes a tedious task the host OS should be installed using a previously created system image if possible.

4.4.2 Software Configuration Management

Now that the operating has been installed on either a virtual or physical machine the next step to begin utilizing infrastructure as code is to install a software configuration management tool. This tool allows for the host machine to be provisioned from a set of configuration instructions. The configuration files describe to the tool what packages to include, how to configure those packages, check out the latest version of a software project, and more. The essential goal of this type of tool is to automate the manual configuration and installation of specific versions and settings of software. The automation ensures each programmer will have an identical development environment assuming everyone uses the same tool and configuration instructions. The three most used software configuration management tools are Puppet [11], Chef [4], and CFEngine [3].
4.4.2.1 Puppet

Puppet was created in 2005 by Luke Kanies [106] and is written in its own configuration language. The configuration scripts are known as modules and their website [11] has previously written open source modules for various desired configurations. The learning curve is manageable after consulting puppet’s very large user community along with the getting started documentation.

4.4.2.2 Chef

Chef was created in 2008 by Adam Jacobs [106] and is written in the Ruby programming language. The configuration packages are known as cookbooks with the individual files being called recipes. The Chef website [4] has a variety of open source community written cookbooks.

4.4.2.3 CFEngine

The first version of CFEngine was created in 1993 by Mark Burgess [106] and is written in its own configuration language. The configuration packages are based on items called promises. A promise is declared and the CFEngine agent ensures that the desired state of the host machine is reached. CFEngine appears to have the highest learning curve, however their website [3] has a helpful documentation section.

4.4.3 Version Control

The infrastructure as code methodology recommends the working infrastructure source code to be checked into a source code repository. This allows members from development as well as operations to have access to the most recent copy. Also, if an error is encountered while updating the infrastructure, the repository allows the infrastructure to be rolled back to a previous version. There are many version control solutions with some of the more widely used options being subversion and git.
4.5 Implementation Use Case

So far this chapter has explained the background, user needs, and requirements of infrastructure as code. With all of this information, it is now possible to begin the process of implementation within an organization. The implementation example described here was achieved within the Center for Research Computing (CRC) at the University of Notre Dame. The CRC is involved with many software focused projects and benefits from this implementation in the traditional ways of decreased errors during employment and improved transition of new employees to utilize the infrastructure as code protocols.

4.5.1 Design Decisions

There are many design decisions that need to be made when deciding to implement the infrastructure as code ideology into a new organization. There were two major design decisions that were made during the implementation at the CRC: which tools to use and whether or not to utilize virtual machines.

4.5.1.1 Software Configuration Management Tools

It needed to be established which software configuration management tool was going to be used. The CRC chose to use Puppet because the syntax was its own language and intuitive where as Chef is written using Ruby and the CRC developers commonly do not use the Ruby language. Puppet also has a large developer community, extensive documentation, and many open source modules. The Puppet client to Puppet master interaction was also very favorable in the design decision. The Puppet master will hold a central repository of all of the Puppet configuration scripts and then the Puppet clients are configured to listen to the Puppet master on how to be provisioned.
4.5.1.2 Virtual Machine Usage

The other design decision was deciding whether virtual machines (VMs) would be used, or if physical machines would be provisioned by Puppet. The solution was a mix of these two options with the developers using virtual machines provisioned by puppet, where the production infrastructure may be physical machines provisioned by puppet. Vagrant was used to manage the VMs from the command line and make creating and destroying these VMs simple. At the time of writing, Vagrant is by default configured to use Virtualbox as the virtualization software.

The virtual machines for the developers were chosen because development is the stage in the software development life cycle where things are most likely to break. By having the developers on a virtual machines, it is easy to restore the VM to the baseline state and mitigate an unnecessary reconfiguration of a broken machine. Additionally, the use of VMs allows for the development environment to be provisioned in such a way that nearly identically matches the production environment.

4.5.2 Configuration

Now all of the design decisions have been made and the tools to used have been selected, they need to be configured in a working way. The tools that were chosen for configuration at the CRC were Vagrant and Puppet, where Puppet consists of both the Puppet client and Puppet master.

4.5.2.1 Vagrant

The first step to correctly configure vagrant is to ensure that the proper dependencies are available. Vagrant requires a base box to have been created and the following command to be issued to allow Vagrant to recognize the base box.

```
vagrant box add 'CentOS_63' 'CentOS_63.box'
```
A base box is a minimal image of an operating system that will serve as a starting point for the system provisioning. There are many publicly available base boxes that users have submitted for use [20].

The next step is to initialize an instance of the previously added base box. The purpose of this step is to tell Vagrant to create the template configuration and settings file that are use to declare how the virtual machine should be booted and provisioned. The following command accomplishes this task.

```
vagrant init CentOS_63
```

A Vagrantfile now exists in the directory where the previous command was issued. The file is responsible for declaring various virtual machine settings including networking, provisioning tool used, hostname, domain name, along with a variety of other options. The Chef or Puppet provisioning tools may be used to provision a Vagrant box. Table 4.1 summarizes the most useful Vagrant commands [36].

### 4.5.2.2 Puppet Client

The puppet client can either be manually installed onto a machine, or if using Vagrant may be specified to be installed and used automatically from within the Vagrantfile. The configuration also may be included in the vagrant basebox created with Veewee to have the correct setting to interact with the CRC Puppet master server. The following command is run by root on the CentOS VM and is used to manually install the puppet client onto a 64 bit CentOS machine.

```
rpm -Uvh http://download.fedora.redhat.com/pub/epel/5Server/x86_64/epel-release-5-3.noarch.rpm
yum install puppet
```

When Puppet is configured to use a client server architecture, the purpose of the puppet client is to periodically check with the server for any updates in its current
TABLE 4.1
COMMON VAGRANT COMMANDS

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>vagrant up</strong></td>
<td>Create a VM from the basebox (or start the VM if it already exists) and provision</td>
</tr>
<tr>
<td><strong>vagrant provision</strong></td>
<td>Reprovision a running VM</td>
</tr>
<tr>
<td><strong>vagrant ssh</strong></td>
<td>Establish a SSH connection to the VM as the vagrant user (default)</td>
</tr>
<tr>
<td><strong>vagrant status</strong></td>
<td>See the status of a VM</td>
</tr>
<tr>
<td><strong>vagrant init</strong></td>
<td>Initializes base box</td>
</tr>
<tr>
<td><strong>vagrant suspend</strong></td>
<td>Same as saving a VM’s state in VirtualBox. Saves the state and closes the VM</td>
</tr>
<tr>
<td><strong>vagrant halt</strong></td>
<td>Shuts the VM down</td>
</tr>
<tr>
<td><strong>vagrant destroy</strong></td>
<td>Destroys a VM image</td>
</tr>
<tr>
<td><strong>vagrant box add</strong></td>
<td>Add a new base box from file or http address</td>
</tr>
<tr>
<td><strong>vagrant box list</strong></td>
<td>List available base boxes</td>
</tr>
<tr>
<td><strong>vagrant box remove</strong></td>
<td>Remove a basebox</td>
</tr>
</tbody>
</table>

configuration. A connection from the puppet client to puppet master only needs to be configured during the initial setup, the subsequent updates are done automatically. To initiate a connection from the client to the master the following command is run by root on the CentOS VM.

```
puppet agent --server puppetmaster --waitforcert 60 --test
```

The term puppetmaster is defined in the /etc/hosts file to be the ip address of the puppetmaster server. The final step is to sign the ssl certificate request by issuing the following commands on the puppetmaster.
At this point everything is properly configured on the puppet client which will check for an update every 30 minutes by default.

4.5.2.3 Puppet Master

The puppet master needs to be initially installed and configured to accept incoming update requests from the puppet clients. The following command is used to install the puppet master onto a 64 bit CentOS machine.

```
rpm -Uvh http://download.fedora.redhat.com/pub/epel/5Server/
x86_64/epel-release-5-3.noarch.rpm
yum install puppet-server
```

To check whether the puppet master has been correctly installed and configured correctly using the default settings the following command is used assuming sudo is setup to have access to puppet.

```
sudo puppet master --mkusers --verbose --no-daemonize
```

If everything is in working order the puppet master will start in the terminal. Assuming sudo is configured, one command needs to be issued in order to daemonize this puppet master process.

```
sudo puppet master
```

To complete the installation, a local test should be executed using sudo assuming it is configured, to verify that the daemon is performing as expected.

```
sudo puppet agent --test --server='hostname'
```

The main unit that is utilized in puppet is called a puppet module. A puppet module is a set of configuration scripts using the puppet scripting language to provision the machine to the appropriate parameters for a given task. For example, there
are puppet modules to configure a machine to be an Apache web server. By default, the puppet modules located on the Puppet Master machine are stored in /etc/puppet/modules/ and the configuration file that specifies which modules are installed on which puppet clients is located at /etc/puppet/manifests/site.pp or /etc/puppet/manifests/nodes.pp. An example site.pp or node.pp file may include a section that provisions the puppet master server itself along with numerous puppet client nodes. The following example file shows how a puppet master would be used to provision the puppetmaster to install the sudo module as well as provision the puppet client to include the Apache module.

```# /etc/puppet/manifests/nodes.pp
define puppetmaster {
  include sudo
}
define puppetclient {
  include apache
}
```

Overall, the client server architecture used in the puppet tool set is very powerful and is shown in Figure 4.3. The benefit of this approach is that it allows for a complete definition of all the nodes along with the server to be stored in one central location which makes updating the configuration of multiple nodes very simple. For example, multiple puppet clients may be provisioned as the same node, so by changing the configuration in the nodes.pp file it will change the configuration across multiple puppet clients during their next update.

4.5.3 Implementation

Now that the infrastructure is set up and everything is configured correctly, the benefits of infrastructure as code may begin to be utilized. Every time the developer wants to start the VM all that is needed to be typed into the terminal is “vagrant up”. This command will start an instance of the previously created CentOS virtual
Figure 4.3. Multiple puppet clients may be apart of the same node and multiple nodes are configured to be provisioned from a puppet master.

machine. Also, Puppet will configure the machine based on the puppet modules that are included with Vagrant. When the developer is done programming the “vagrant halt” or “vagrant destroy” commands are issued which put the VM to sleep or destroy the VM respectively.

The puppet language separates various packages and software implementations into individual modules. When a new project is started it is the responsibility of the developer to establish the initial infrastructure by combining the individual puppet modules that are required. For example, if an Apache web server with firewall port 80 open and java support is required, the Apache, firewall, and java puppet modules will be included in the puppet manifest. Multiple puppet modules were manually created,
altered from an open source version, or directly used for the CRC are summarized in Table 4.2. Full source code of select puppet modules are available in Appendix E.

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>HTTP server</td>
</tr>
<tr>
<td>Celery</td>
<td>Asynchronous task queue/job queue based on distributed message passing.</td>
</tr>
<tr>
<td>Compute Node</td>
<td>Base CentOS machine image</td>
</tr>
<tr>
<td>Fedora Commons</td>
<td>A general-purpose, open-source digital object repository system</td>
</tr>
<tr>
<td>Jenkins</td>
<td>An open-source continuous integration server with 300+ plugins to support all kinds of software development</td>
</tr>
<tr>
<td>Lettuce</td>
<td>Behavior Driven Development tool used to perform software testing</td>
</tr>
<tr>
<td>MongoDB</td>
<td>An open source document-oriented database system</td>
</tr>
<tr>
<td>mySQL</td>
<td>An open source relational database management system</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>A object-relational database management system available for many platforms</td>
</tr>
<tr>
<td>Puppet Server</td>
<td>A server used to manage puppet clients through a client-server architecture</td>
</tr>
<tr>
<td>RabbitMQ</td>
<td>An open source message broker software that implements the Advanced Message Queuing Protocol (AMQP) standard</td>
</tr>
<tr>
<td>Redis</td>
<td>An open source, BSD licensed, advanced key-value store</td>
</tr>
<tr>
<td>Tomcat</td>
<td>An open source web server and servlet container</td>
</tr>
</tbody>
</table>
In addition to the puppet provisioning, the VM may check out the latest version of source code from a source code repository. This allows the developer to begin working right away and ensures that the latest version of the software is always loaded into the VM upon creation. To ensure this is working the developer will need to checkout the source code to the desired directory with the necessary credentials. Once this is complete, this process may be automated for future instances of the VM.

4.5.4 Encountered Issues

The most common issue encountered during the implementation had to do with interfacing with the network. The VMs could be configured through Vagrant to either connect to the network using a host-only connection or through a network bridge. The host-only connection caused the VM to only be seen by the host computer, but could be accessed by other computers on the network through port forwarding from the host to the VM. The ideal solution was to have the VM register with the DNS of the network and be assigned its own IP address. However, this approach did not always work using Vagrant and the host-only with port forwarding solution was pursued.

Another issue that was encountered appeared when accessing the open source puppet modules. The CRC had chosen to use a CentOS 6.3 image for their virtual machine base and sometimes the previously published puppet module was not written to accommodate this operating system. The puppet module was then manually modified to provide the appropriate provisioning of the CentOS VM. The issue deals with when a puppet module was created from scratch whether or not to take the extra time to make it cross platform compatible. Currently, all of CRC’s development VMs are CentOS, but if in the future they decide to switch the base OS the issue may arise again where they will have to modify certain puppet modules to correctly provision the new OS.
4.6 Limitations and Future Work

A limitation of the infrastructure as code approach to improve software development and system administration is the assumption that puppet will be used in conjunction with a Unix operating system. There are documented ways on how to install a Puppet client on a Microsoft Windows workstation, but all of the Puppet modules described in this thesis have been written for a CentOS operating system. Additionally, a majority of the published Puppet modules are also for Unix operating systems. Another limitation is that if Microsoft Windows workstations need to be configured through a Puppet client to Puppet master configuration, a Unix server must be present because a Microsoft Windows node may not function as a Puppet master server.

The work from this chapter has looked at the history and an implementation case study of infrastructure as code. A future direction for research that should be pursued is to quantify the results of implementing this methodology within the CRC. The literature states that this approach should improve efficiency and reduce errors, but a study should be performed on the impact infrastructure on code had on the development operations of the CRC.

Another future direction for research is to perform a more in depth analysis of the software configuration management tools and clearly identify situations or environments where certain tools may perform better than the others. In the example implementation, the CRC chose to use Puppet because of their current limited level of exposure with the ruby language, however it may be possible that it would be more beneficial to go with another software configuration management tool due to potential functionality needs that are not being met by Puppet. At the time of writing this chapter, Puppet accomplished all of the requirements determined by the CRC. However with the field constantly changing, the CRC requirements may change as well and they may need to reassess their design decision to use puppet over the other
software configuration management tools in the future.

4.7 Summary

DevOps, more specifically infrastructure as code, is shaping the current software development industry. This methodology helps reduce errors during the deployment of software as well as reduce the complexity of training new employees to work on projects. By defining the user needs and the implementation requirements of an organization, the implementation of infrastructure becomes not only a possibility but a reality.
CHAPTER 5

CONCLUSION

This thesis discussed the overall concepts, implementation details, and future directions of two modern computing paradigms: mobile computing and infrastructure as code.

5.1 Haiti Infrastructure Surveyor Mobile Application

The HIS mobile application has the potential to improve the safety and overall quality of life for communities all over Haiti. Just ten years ago, the concept of crowd sourcing or citizen engineering was just beginning and the direct application of these new data obtainment methods was still unclear. Today, there exists dozens of examples of projects that have benefited from a crowd sourcing approach. The HIS application has the potential to help the citizens of Haiti take an active role in improving their safety and the structural stability of buildings within their community.

5.2 Infrastructure as Code

DevOps and infrastructure as code both are shaping the way software development is being accomplished today. Companies and organizations including Google, Mozilla, and Harvard Law School [33] are beginning to change their approach to developing software by incorporating the infrastructure as code design method into their daily practices. With the proper expertise and resource investment, the benefits
of a DevOps approach may outweigh the initial setup cost. Overall, the DevOps and specifically infrastructure as code approach to software development is transforming the industry and will greatly improve the time to deployment, training process of new personnel, and overall collaboration among teams.
This document was prepared by Aaron Huus and Ryan Michaels in CSE 60232 Software Engineering on 9/28/2012 which was instructed by Dr. Collin McMillan.

A.1 Introduction

A.1.1 Product Purpose

The main purpose of this application is to facilitate the collection of demographic and structural data about local infrastructure from a community member with minimal training. The Haiti infrastructure android application will eventually be placed on 100 smart phones and distributed within Haiti. Each phone recipient will be instructed to document and photograph crumbling infrastructure within their community utilizing the smartphone application.

A.1.2 Product Scope

The android application will facilitate the use of crowd sourcing to gather structural data about crumbling infrastructure in third world countries. A benefit of using this approach is that it allows for large amounts of data to be gathered over a short period of time. All of the data captured from this application will be sent to a central database in which researchers will have direct access allowing for structural integrity simulations to be performed. The application is only part of a larger system that will
need to be implemented simultaneously in order to maintain the functionality of the current application.

A.1.3 Product Prospective

The application is created as a subcomponent of a larger system which includes a backend database, GIS visualization tools, structural integrity simulation models, and a risk reporting application. The GIS visualization tools and structural integrity simulation models heavily rely on the data that is captured and stored in the mySQL database from the android mobile application.

A.1.4 Product Functions

In order for our application to be successful it must be able to perform the following tasks as described in an interview with Dr. Madey of the computer science department at Notre Dame and Dr. Kijewski-Correa of the civil engineering department at Notre Dame:

- Have a wizard direct the user through the data collection within the application
- Prompt the user with demographic questions pertaining to the current structure
- Document any observations
- Take a photo
- Record the GPS location of each photo taken
- Upload all of the captured data to a remote database
A.2 Interface Requirements

A.2.1 User Interfaces

The user interface will be a simple to use GUI (Graphical User Interface), direct on the android phone. The application has several steps, and the user will be guided through each one by the Wizard.

A.2.1.1 Parcel Data

The first thing the user sees when they open the application is a page for entering demographic information, which will contain several text fields. The user is prompted to enter all relevant information they can for the background information to be complete. This information can range from highly specific, such as exact street name and number, to general, for example the home is located next to the 3 pomegranate trees.

A.2.1.2 Document Observations

After entering the parcel data the user is prompted to begin entering any observation about the structure itself. This will be accomplished once again through text field entry.

A.2.1.3 Photo

At any point during this process the user may take one or more photographs with the phone. The camera will be activated by pushing a photo button in the bottom of the GUI. This brings up the camera on the android phone. The user will then take one or more photographs of the region of interest. They will be given the option to save or discard each photo once a DONE button is pressed. If the user chooses to keep a photo it is sent to the database and the phone automatically attaches GPS
coordinates to the photograph. The user is also given the ability to add comments, or tags, to the photo if desired.

A.2.1.4 Report

At any point the user may send data (both textual information and photographs) back to the central server, provided that the phone is connected to the internet. This will send the data collected up to that point for the project back to the central database, where it will be processed and stored.

A.2.2 Hardware Interfaces

The hardware utilized will be the Motorola XT720 mobile smartphone.

A.2.3 Software Interfaces

The software will utilize text entry, photo taking, geotagging, and information transfer. The text will be saved to a temporary file which will be deleted upon its successful transfer to the database.

A.2.4 Communication Interfaces

This application will have one way communication with a remote SQL database. It will therefore require an internet connection for the duration of communication. The information will not be encrypted at this stage of early development, this feature may be added at a future date.
A.3 System Features

A.3.1 Wizard Walkthrough

A.3.1.1 Description

In order to keep the application intuitive and simple, a wizard guides the user through the data acquisition process. A wizard is a series of questions or prompts that will provide instructions to the user on what the next step in the data acquisition process is.

A.3.1.2 Stimulus/Response Sequences

The user is required to respond to the action or question that has been prompted by the wizard and the wizard will not continue until that action item has been completed. These action items consist of answering a question, taking a photo, or documenting observations.

A.3.1.3 Feature Rationale

The consulting professors indicated during an interview that it was very important that the application be used with very minimal training. The idea of a wizard that prompted the user through the necessary stages of data acquisition was determined to be the best way of accomplishing this.

A.3.1.4 Functional Requirements

- R1 - Workflow:

  The workflow of the wizard always start with asking the user to answer a general questionnaire about the structure to obtain as much of the general history as the user knows. The wizard then proceeds to prompt the user to take photos of all the exterior sides of the structure, take observational notes about the
exterior, take photos of all the interior walls, and record observational notes about the interior.

- R2 - Concise Instructions:
  The wizard provides short and specific instructions on exactly what the user should do. These instructions are never longer than 2 sentences and they never ask the user to do more than one task.

A.3.2 Parcel Data Questionnaire

A.3.2.1 Description

Parcel data consists of basic demographic questions that gather data related to the current structure being examined. The questionnaire tries to acquire a greater understanding of the history and location of the structure by asking questions such as when the house was built, who built it, and what is the street address or intersection of the structure.

A.3.2.2 Stimulus/Response Sequences

The user is prompted to enter basic demographic information about the structure of interest, including: the buildings age, nature of the surroundings, address, material(s) the building is constructed out of, and number of occupants. This prompting will be done in the form of several standardized questions. The user will be allowed to enter in a response for unknown or missing information.

A.3.2.3 Feature Rationale

This information is important for building an understanding of the background and history of an area. This will allow for higher level analysis to potentially identify wide areas which need assistance, for example if twenty houses in a neighborhood are
all constructed out of materials very resistant to an earthquake, but are vulnerable to a hurricane ravaging them, this information would allow the volunteers to notify whole communities on the potential risk(s).

A.3.2.4 Functional Requirements

• R1 - Visual Display:
  Each question is displayed to the user via an interface that allows the user to input comments via a text field under the provided questions.

• R2 - Question Content:
  The questions are based directly on the data needed as provided by Dr. Kijewski-Correa.

A.3.3 Document Observational Notes

A.3.3.1 Description

While observing the location and overall structural stability of the unit, the wizard provides a text box where general observational notes about the interior/exterior may be recorded.

A.3.3.2 Stimulus/Response Sequences

Much like the above parcel data capture, the Observational Notes interface will provide a way for the user to input their observations about the structure of interest. Data will be entered into several fields including: Exterior concerns, interior concerns, surrounding area concerns.
A.3.3.3 Feature Rationale

This will be used to document any features which the user feels are relevant for consideration when considering the safety of the structure, especially for any information which may not be able to be captured by the camera on a phone, such as a fine layer of cracks in the wall, the floorboards being weakened by constant flooding, or material shortage for repairs on the structure.

A.3.3.4 Functional Requirements

- R1 - Text Prompt:
  This feature will be provided via a standard text interface. It will be guided by pop up messages with a text field for answering supplied.

A.3.4 Take a Photograph

A.3.4.1 Description

One of the major requirements of this application is to have the user take pictures of the interior and exterior of the structure being examined. The photographs are a necessity for the research scientists who will later observe these pictures to try and assess the structural stability of the building.

A.3.4.2 Stimulus/Response Sequences

The user, either upon completing the parcel data collection and the observational notes or at any time during that process, will be allowed and then prompted to take photos of the structure in question. Specifically one (or more) of each side of the structure, and then several of the interior of the building. The user will geotag these photos if the phone itself does not automatically provide this functionality.
A.3.4.3 Feature Rationale

In Haiti parcel data is nonexistent, so by taking pictures of the structure some information may be extrapolated. The pictures provides the research scientist with a better understanding of the overall scene of the structure being observed.

A.3.4.4 Functional Requirements

- R1 - Geotag Photograph:
  The photos must each be geotagged with the GPS coordinates of the image location so that the analysts who will be looking at the image can cross reference any concerns that the area may have present with, such as proximity to coastline, distance from major fault lines, and distance from volcanic centers, in order to give the most succinct and relevant information on what to do to keep the structure protected.

- R2 - Space availability:
  The photos must be temporarily stored locally on the phone, and there will be an option for deletion both after taking the photo and after it has been verified that the database has received the photo. This application will tag each of the photos with an identifier so that in the event of the phones memory being full, the application knows which photos it may delete. It will follow a policy of deleting the oldest photo first. This will prevent the application from deleting the users personal photos. If after deleting all photos taken by the application itself there is no room, the user will be informed that there is no more space available on their phone for this applications use, and to please clear some memory.

- R3 - Temporarily Photograph Storage:
  The photos are stored locally on the phone until it may be confirmed that they
have been uploaded to the database.

A.3.5 Database Server Communication

A.3.5.1 Description

While the observational notes and photographs of a particular structure are taken they need to be uploaded to a remote database. This is accomplished through the use of a cellular signal to gain access to the internet, and the correct syntax to interact with the mySQL database on the backend.

A.3.5.2 Stimulus/Response Sequences

At any point once the user has started data entry he will have the option to report back to the database. This will be accomplished via a button in the bottom of the GUI. Once this button is pressed, assuming there is an internet connection present, all information that the user has for the current project is sent off to the database. Once this is complete the user is prompted to continue entering information or to close the current project.

A.3.5.3 Feature Rationale

For the back end scientists to be able to use the data being gathered there needs to be a centralized storage area for all of the data, otherwise it would be scattered about, non localized, and have a high chance of data being corrupted in some form or another.

A.3.5.4 Functional Requirements

• R1 - Upload Verification:

  The application waits for an upload verification response from the database to
ensure that the data has successfully been transferred and the local copy is no longer needed.

- R2 - mySQL Database:

  The commands that are sent to the database for uploading the photos and observational notes comply with current communication strategies for a mySQL database.

A.4 Nonfunctional Requirements

A.4.1 Performance

  The application must react to the user’s input in a perceived instantaneous fashion so as to allow for the user to interact with the application in a natural fashion. A status bar is used to notify the user that the application is working on a task when it cannot complete instantaneously. Also, the application must load within 8 seconds of clicking on the icon. This ensures that the user does not think the application is frozen and reinforces the notion of allowing the user to interact with the application in a natural fashion.

A.4.2 Security

  While communicating with the database server a username and password is used to authenticate with the server. This prevents unauthorized users from obtaining or committing files to the database. Another security measure that is taken is to provide each entry sent to the database will be tagged with a unique identifier. This is another verification that the data being committed is valid and also may be used to track the individual progress of each community member using the application.
A.4.3 Availability

The application will be made available by providing a private download link pointing to the .apk file of the Haiti infrastructure application. The file will install the application onto the mobile device assuming this device is running a version of the android operating system.

A.4.4 Maintainability

The application will be maintained from within the development environment as described in the development environment section of this document.

A.4.5 Feature Testability

A.4.5.1 Wizard Walkthrough

The wizard is tested by having an individual use the application and see if the wizard adequately leads the user through the data acquisition process. Also, the tester will verify that all of the question are concise questions and only address one issue per question.

A.4.5.2 Data Questionnaire

The questionnaire is tested by having Dr. Kijewski-Correa confirm the appropriateness of the questions being used. Also, the question display may be observed by the tester to verify that it is conducive to obtaining the parcel data.

A.4.5.3 Document Observational Notes

The tester may note the use of a pop-up message with a text field to allow the user to input any observational notes.
A.4.5.4 Take a Photograph

The tester can verify that the pictures are geotagged by looking at one of the uploaded photos. Also, if the tester tries to take a picture when there is not enough space the error message prompt may be observed. Finally, if the tester takes a picture while there is no internet connection, it is verified that the picture is uploaded once internet connectivity is reattained.

A.4.5.5 Database Server Communication

The tester may observe the source code to verify that an upload verification is obtained before deleting the photo. Also, the communication protocol between the android application and the database is easily verified to be a form of SQL.

A.4.6 Traceability

The application code will utilize comments to clearly describe which of the requirements as described in this software requirements specification document are being implemented. There will also be a separate document in the source code that precisely links the stated document requirements to where it is implemented within the source code.

A.4.7 Development Environment

The application will be developed using the android SDK and the Eclipse IDE and written in the Java programming language. It will be developed for android 4.1 Jelly Bean and will be backwards compatible with previous versions of the android operating system. The Motorola XT720 mobile smartphone will be used for development and test deployment.
A.4.8 Operating Environment

The Haiti infrastructure android application it to be installed on a mobile device running the android operating system. The mobile phone must have a functioning camera, hardware or on-screen keyboard, GPS, and internet connectivity.

A.5 Other Requirements

A.5.1 User Documentation

An easy to follow user manual written in Creole will be written by the deployment team and made available to the community individuals who are utilizing the application. The document will help summarize the responsibilities required of the community member as well as instruct on how to perform some of the basic functions such as accessing the application, taking a photo, and documenting observations.

A.5.2 Assumptions and Dependencies

A major assumption that is made that directly impacts the functionality of the application is the availability of either a 3G or wireless internet signal. Without any form of internet connectivity the proposed crowd sourcing solution is ultimately ineffective. It has been confirmed by both researchers as well as community members that there is indeed a way to access the internet from a mobile device.

The application depends on a functioning backend database which is being developed by the EPICS team at Notre Dame. The database stores the pictures and documentation notes gathered by the community individual. Without a database, the photos and notes will be stored locally on the phone until they are able to be uploaded and therefore may not be accessed by the research scientists interested in this data.
A.5.3 Future Enhancements

A.5.3.1 Compass Utilization

Any the time that the user takes a picture the compass is accessed to determine the cardinal direction that the user is facing at this time. Knowing this information will help the research scientists better conceptualize the location and orientation of the structure.

A.5.3.2 Automatically Obtaining Dimensions

A known object with known dimensions is placed in each picture the user takes. The purpose of this object is to allow for the research scientists to later extrapolate the dimensions of the wall or structure using the known object as a reference point. This would allow for a more accurate depiction of the structure to be obtained.

A.5.3.3 Floor Plan Sketching

Currently the floor plan is described textually through a series of questions and a comment box within the application. It has been expressed that it would be useful if part of the application would allow the user to draw the floorplan of the structure. The drawn floor plan would be saved as an image file and uploaded along with the other data obtained.
B.1 Design Overview

The Haiti android application uses the structured design method. The application has been split into five top level modules: structure selection, questionnaire, exterior observations, interior observations, and data submission. Each of these modules communicates with each other in a wizard fashion such that the module elicits a call to the subsequent module to take over directing the users actions.

The structure selection module is responsible for maintaining a list view of previously started but not submitted structure surveys and is where the user will always begin interacting with the application. The questionnaire module appears once the structure selection module identifies which structure the user will be working with. Upon finishing this module, the user is directed to the exterior observation module. It is here where the user beings to take photos of the exterior of the structure and document various observations. The interior module is called on after the completion of the exterior module. This module functions similar to the exterior module in that it requires the user to document the interior of the structure using textual and pictorial observations. After completion of the interior module, the data submission
module is called to assist the user in committing the recorded observations to the
database.

B.2 Architectural Model

This application follows a very clear path of execution, with one module running
in its entirety before proceeding to the next module (Figure B.1). The modules
are as follows: Structure selection, Questionnaire, Exterior Observations, Interior
Observations, and Data Submissions. Each module's functionality is described below
in brief, and echoed in Figure B.1.

- Structure Select: Allows creation of a new project, or loading of an incomplete
analysis.

- Questionnaire: Collects parcel data about the structure. Saves to local memory.

- Exterior Observations: Captures an image for each outward wall of the struc-
ture. Allows for documentation of any exterior issues [materials, damages, and
so forth]. The images are placed on a diagram of the home to designate location.
The results are saved to local memory.

- Interior Observations: Maps the floorplan of the structure, and allows for doc-
umentation of any interior issues [materials, structural damages, leaks, and so
forth]. The results are saved to local memory.

- Data Submission: Submits the saved data from the previous three modules to
the remote server.

Figure B.2 provides a high level view of the Exterior Observations Module.

B.3 Module Overview

The modules are summarized in the following table:
B.3.1 Structure Selection Module

B.3.1.1 Module Behavior

The Structure Selection modules purpose is to provide the user with an interface that allows him/her to select the structure that is currently being observed. A list view is loaded that displays the currently open structures. If no structures have been started in the past, then there is an option that allows the user to initiate an instance of a structure that will then populate the list. When a structure from the list is selected the questionnaire module is started.
Figure B.2. Summary of the exterior observations module where dotted circles represent individual images and boxes represent textual observations. The solid circle represents the central structure being observed which each image and textual observation are correlated with.

B.3.1.2 Assumptions

This module makes the assumption that there exists a database in the background of the application that keeps track of the open structures and is able to add a new structure to this database.

B.3.1.3 Secrets

The list view may only be seen and interacted with from within the scope of the structure selection module.
TABLE B.1

MODULE OVERVIEW

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structure Selection</td>
<td>Allows the user to add or select a structure to begin documenting</td>
</tr>
<tr>
<td>Questionnaire</td>
<td>Gathers basic information about the structure being observed</td>
</tr>
<tr>
<td>Exterior Observations</td>
<td>Directs the user through the process of documenting the exterior</td>
</tr>
<tr>
<td>Interior Observations</td>
<td>Directs the user through the process of documenting the interior</td>
</tr>
<tr>
<td>Data Submission</td>
<td>Collects the acquired data about the structure and submits it to a backend database</td>
</tr>
</tbody>
</table>

B.3.1.4 Constraints and Limitations

The module is limited in the length of the name given to the structure. The reason for this limitation is to allow for the entire name to display on one line within the list view. The exact size of the name field will be determined programmatically. There are no size restraints with regards to documentation comments throughout the application.

B.3.1.5 Error Handling

If the name of the structure is too long, the user will be prompted to shorten the length of the name. Also, if an error occurs while grabbing the list from the database an error message will be displayed to the user as to why this issue has occurred.

B.3.1.6 Expected Changes

An expected change that may be implemented for this module is how the structure list is displayed. Right now the list view makes the most sense, but if the application was ever to be ported to a tablet interface alternative display methods may be better suited for this task.
B.3.1.7 Related Requirements

3.1.4.R1 and 3.1.4.R2 are met which states the wizard must state concise instructions and addresses the overall workflow of the application respectively.

B.3.2 Questionnaire Module

B.3.2.1 Module Behavior

The questionnaire modules purpose is to prompt the user with questions about the structure being surveyed. Related questions are presented on the same page, and each view will only have one button to advance to the next page of questions. The final page of questions has a button that will advance the user to the exterior observation module.

B.3.2.2 Assumptions

An assumption that this module makes is that the user knows the answer to all of the questions that require a response. An issue may arise where the user does not know the answer to one of the questions and therefore cannot move on. We plan to select the required questions in such a way that this occurrence will rarely happen.

B.3.2.3 Secrets

The questions and questionnaire interface may only be called from within the questionnaire module.

B.3.2.4 Constraints and Limitations

The file that is created by saving the responses to all of the questions must be able to be stored on the internal memory of the phone or if the mobile device has a SD card it may also be stored there.
B.3.2.5 Error Handling

If a certain question requires the answer to be in a specific format such as a phone number, there will be an error message that will prompt the user to input the answer with the correct format and an example will be shown.

B.3.2.6 Expected Changes

An expected change that may be implemented for this module is how the questions are clustered together. As a user begins to use this application, it may be found out that certain questions should be clustered together on one screen view. This will be easily altered by changing the layout xml file that holds the interface.

B.3.2.7 Related Requirements

3.1.4.R1, 3.1.4.R2, 3.2.4.R1 and 3.2.4.R2 are met which address the wizard workflow, concise questions, question display, and question content respectively.

B.3.3 Exterior Observations Module

B.3.3.1 Module Behavior

The purpose of the exterior observations module is to guide the user through the process of collecting data about the exterior of the structure. A pictorial representation of the exterior of the structure is shown to the user with an arrow that indicates which wall of the structure is being documented. A photo and note about each side of the exterior are documented and written to a file.

B.3.3.2 Assumptions

This module assumes that there is storage capacity on the device for collected data to be stored upon, as well as an external database which will receive the data.
The module also assumes the device has a camera, and that the camera is functional.

B.3.3.3 Secrets

The Exterior Observation interface may only be called via the Exterior Observation Module. The exterior mapping will only be done via the Exterior Observation module.

B.3.3.4 Constraints and Limitations

This module will take a minimum of four pictures, as well as obtain some textual data capture. There must be enough free space on the mobile device for all the obtained photographs and observations to be stored. Photographs taken must be geotagged with the correct latitude and longitude coordinates.

B.3.3.5 Error Handling

If the user attempts to move on without placing a taken picture within the diagram of the structure an error will be thrown, and the user will not be able to move on, either with taking another picture or with proceeding to the next module depending on the number of pictures taken, without placing the current picture.

B.3.3.6 Expected Changes

The major change that is planned on is to allow a more variable exterior structure to be captured, for example a house that has a non standard number of exterior walls, or a collection of homes in a slum type manner [where each house is connected to those adjacent to it, buildings may be stacked atop one another.]
B.3.3.7 Related Requirements

3.1.4.R1, 3.1.4.R2, 3.3.4.R1, 3.4.4.R1, 3.4.4.R2, and 3.4.4.R3 are met which address the wizard workflow, concise questions, text prompt, geotagged photographs, space availability, and temporary photograph storage respectively.

B.3.4 Interior Observations Module

B.3.4.1 Module Behavior

The purpose of the interior observation module is to guide the user through the process of collecting data from the interior of the structure. The user will document with photographs and notes for each room of the structure. This module also helps capture the floor plan of the structure through a series of questions and actions that the user executes and records the results. The data acquired from this module is appended to the end of the file previously.

B.3.4.2 Assumptions

This module assumes that the previous two modules have been run to completion. It assumes that the mobile device has enough storage capacity to hold the information collected. The module assumes that the user will be able to capture the basic house floorplan, and several architecturally relevant details through access to the roof.

B.3.4.3 Secrets

The image capture for this module will only be called within the Interior Observations Module. Observations about the structures interior will only be captured by this module.
B.3.4.4 Constraints and Limitations

This module will capture textual and photographic data, both to document the floorplan and the conditions of the home. The mobile device must have enough storage capacity to hold all the captured images. If the user is unable to capture images from the roof, rough approximations may be entered, and the file designated as incomplete/too dangerous to complete.

B.3.4.5 Error Handling

If the user attempts to proceed without capturing a floorplan, an error will be thrown. The user must either report the floorplan as too dangerous to capture, or capture a floorplan to continue. If text data is entered in an unrecognized format (emoticons for example) an error will be thrown.

B.3.4.6 Expected Changes

The expected change is a design that makes the floorplan capture more efficient or less dangerous for the user.

B.3.4.7 Related Requirements

3.1.4.R1, 3.1.4.R2, 3.3.4.R1, 3.4.4.R1, 3.4.4.R2, and 3.4.4.R3 are met which address the wizard workflow, concise questions, text prompt, space availability, and temporary photograph storage respectively.

B.3.5 Data Submission Module

B.3.5.1 Module Behavior

The purpose of the data submission module is to send the data file created from the previous modules to a backend database server. The previous modules wrote the
acquired information to a file, and this module will transmit the file to the designated server. It will then be the responsibility of the server to parse the needed data from the file.

B.3.5.2 Assumptions

The module assumes that there is any available internet connection (WiFi or 3G) with enough signal strength to send the data file to the central server. The module assumes that the data file being sent contains all data that was gathered by the above modules. It assumes that the central server is functioning and able to both receive and process the data.

B.3.5.3 Secrets

The module will send all data collected by the previous three modules to a remote database. The Data Submission Model has no way to extract data from the database, only to send data.

B.3.5.4 Constraints and Limitations

This module can only send files saved by the previous modules. All three previous modules must have been completed for this module to be activated. There must be a consistent internet connection for the module to begin sending of data.

B.3.5.5 Error Handling

If all the sent files are not received by the database an error will be sent to the mobile device requesting a resend of the file(s) in question. Disruptions of internet service will produce an error, and require the resending of all incomplete data.
B.3.5.6  Expected Changes

It is expected that the connection to the database may evolve over time, structure of the saved files may change to be more efficiently loaded into the database.

B.3.5.7  Related Requirements

3.5.4.R1 and 3.5.4.R2 are met which address upload verification and the mySQL database respectively.

B.4  Use Relationships

Table B.2 indicates the uses relationships among the Haiti android application modules.

<table>
<thead>
<tr>
<th>Module</th>
<th>Uses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structure Selection</td>
<td>N/A</td>
</tr>
<tr>
<td>Questionnaire</td>
<td>Structure Selection</td>
</tr>
<tr>
<td>Exterior Observations</td>
<td>Questionnaire</td>
</tr>
<tr>
<td>Interior Observations</td>
<td>Exterior Observations</td>
</tr>
<tr>
<td>Data Submission</td>
<td>Interior Observations</td>
</tr>
</tbody>
</table>
B.5 Rationale

The Haiti android application required various design decisions to be made throughout the implementation process. These decisions are summarized here:

- The structure selection module needed a way to display the structure that had been started. One way this could have been done was to display the structures in progress by creating a new button for each structure. This process was quickly discredited because of the difficulty involved with dynamically creating an unknown number of buttons at one time. We then thought about what we really needed from this module, and that turned out to be a simple list. Within the android environment a list view may be created and certain instances (i.e. structures) may be initiated. With this functionality a separate object within the list is created for each structure in progress which is exactly what was desired.

- The questionnaire modules main purpose is to ask the user questions about the structure being observed. The original implementation was to have pop-up messages prompt the user with the various desired questions. An issue we saw with this approach was that a pop-up message may be dismissed by the user as annoying and will never actually get read. With this in mind, we decided to pursue an approach where a series of questions are displayed to the user by changing the view. This avoids the issue of it being a pop-up because the questions consume the entire screen. The screen will only have one button, either next or save, and some of the questions are required. Therefore, the layout design change is very simple and avoids the user disregarding questions due to the nature of them being presented as a pop-up.

- The exterior and interior modules needed a way to walk the user through the data acquisition steps. One option we thought about pursuing was to use a
textual guidance scheme similar to the questionnaire module. However, we quickly realized that it may become very complicated to describe to the user where he/she was in the overall process. Therefore, we decided to pursue an interface that involved a pictorial representation. The picture will greatly help the user understand what task they are currently accomplishing along with how it relates with the overall progress of the structural survey.

• The data submission module needed a decision made as to how the application would communicate with the backend server. We initially thought about communicating with the SQL server directly. This was disregarded because we wanted our application to be functional with any version of backend server and not be limited to a SQL database. This lead us to pursuing the option of writing the data collected to a file. The file will then be uploaded to any backend server which will be responsible for parsing out the data from our file and populating the database.
APPENDIX C

GETTING STARTED WITH ANDROID MOBILE APPLICATION DEVELOPMENT - HIS APPLICATION

C.1 Getting Started Documentation

The best place to get started is to begin looking through the documentation provided on the Android Developers website. The following website clearly describes how to design, develop, and distribute your first android mobile application:

http://developer.android.com/about/start.html

C.2 Software and Tools Needed

An overall guide on how to install the following tools and get started with android development may be found at:

http://www.androidhive.info/2011/05/get-started-developing-for-android-with-eclipse/

C.2.1 Android SDK

The Android SDK is a very important tool that provides API libraries, and the developer tools needed to build, test, and debug applications. The SDK may be downloaded at:

C.2.2 Eclipse

Eclipse is the most common IDE used for Android mobile application development and may be downloaded from:

http://www.eclipse.org/downloads/

C.2.3 Eclipse ADT Plugin

From within Eclipse install the developer tools associated with this location by clicking on Help and then Install New Software. Enter in the following information:

Name: Android
Location: https://dl-ssl.google.com/android/eclipse/

C.3 First Application

Now the development environment should be set up and you are now ready to start creating an Android mobile application. The following Android developers website provides a wide array of very helpful tutorials on how to get started implementing various features:

http://developer.android.com/training/basics/firstapp/index.html

C.4 Source Code

Appendix D has more details on the specific source code which may be found in its entirety in a google code git repository located at:

http://code.google.com/p/ndepics-haiti-recovery/
APPENDIX D

HIS Mobile Application Source Code

A sample of the source code for the HIS application is included through the following sections. Section 1 includes the main activity, Section 2 includes questionnaire activity page 1, Section 3 includes the exterior activity. Various helper and utility classes along with the full view of the source code may be accessed at the time of writing via the SVN repository at:
http://code.google.com/p/ndepics-haiti-recovery/

D.1 Main Activity

```java
package com.example.haiti.infrastructure.surveyor;

import android.app.AlertDialog;
import android.app.Dialog;
import android.app.ListActivity;
import android.app.ProgressDialog;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.os.AsyncTask;
import android.os.Bundle;
import android.util.Log;
import android.view.ContextMenu;
import android.view.MenuItem;
import android.view.KeyEvent;
import android.view.LayoutInflater;
import android.view.MenuInflater;
import android.view.MenuItem;
```
public class main_Activity extends ListActivity {

   private Context mContext;
   private ListView mainListView = null;
   CustomSqlCursorAdapter adapter = null;
   private Cursor currentCursor = null;
   private Utils utilsHelper = null;

   private int structureID;

   private ListView listView = null;

   private static final int STRUCTURE_ADD_DIALOG_ID = 0;
   private static final int STRUCTURE_EDIT_DIALOG_ID = 1;

   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.structure_list_view);

       // Captures application activity to the mContext
       // variable
       mContext = this;

       // Creates the dbHelper object and opens the database
       utilsHelper = new Utils(mContext);

       setupAddStructureButton();

       new SelectDataTask().execute();

       setupListView();

   } // end of the onCreate method
private void setupListView()
{
    listView = getListView();
    listView.setItemsCanFocus(false);
    listView.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE);

    this.mainListView = getListView();
    mainListView.setCacheColorHint(0);
    registerForContextMenu(getListView());
}

private void setupAddStructureButton()
{
    // Add structure button
    ImageButton btnAddStructure = (ImageButton)
        findViewById(R.id.btnAddStructure);
    btnAddStructure.setOnClickListener(new OnClickListener()
    {
        public void onClick(View v) {
            showDialog(STRUCTURE_ADD_DIALOG_ID);
        }
    });
}

@Override
protected void onRestart()
{
    super.onRestart();
    new SelectDataTask().execute();
}

@Override
protected void onPause()
{
    super.onPause();
    utilHelper.closeDatabase();
}

@Override
protected void onStop()
{
    super.onStop();
    utilHelper.closeDatabase();
}

private class SelectDataTask extends AsyncTask<Void, Void, String>
{
    private ProgressDialog pdia;
}
@Override
protected void onPreExecute(){
    pdia = new ProgressDialog(mContext);
    pdia.setMessage("Loading Structures ...");
    pdia.show();
}

protected String doInBackground(Void... params) {
    try {
        utilsHelper.openDatabase();
        main_Activity.this.currentCursor = utilsHelper.
dbHelper.getCursor(null);
    } catch (SQLException sqle) {
        throw sqle;
    }
    return null;
}

protected void onPostExecute(final String result) {
    pdia.dismiss();
    startManagingCursor(main_Activity.this.
currentCursor);
    int[] listFields = new int[] { R.id.
structure_title };
    String[] dbColumns = new String[] { SqlHelper.
COLUMN_STRUCTURE_NAME };

    main_Activity.this.adapter = new
CustomSqlCursorAdapter(
    main_Activity.this, R.layout.
structure_item,
    main_Activity.this.currentCursor,
    dbHelper, listFields,
    utilsHelper.dbHelper);
    setListAdapter(main_Activity.this.
adapter);
}

@Override
protected void onListItemClick(ListView l, View v, int position, long id) {
    // TODO Auto-generated method stub
    super.onListItemClick(l, v, position, id);

    structureID = (int)id;
    utilsHelper.startNextActivity(structureID, questionnaire_Activity1.class);
}

// A pop-up dialog is created to allow the user to create and edit a structure.
// REQ 3.3.4.R1 met
@override
protected Dialog onCreateDialog(int id) {

    switch (id) {

    case STRUCTURE_ADD_DIALOG_ID:

        AlertDialog.Builder builder = new AlertDialog.Builder(this);

        // Inflates the custom xml file layout
        create_structure_dialog.xml
        LayoutInflater inflater = this.
        getLayoutInflater();
        View dialogView = inflater.inflate(R.layout.
        create_structure_dialog, null);
        builder.setView(dialogView);

        builder.setTitle(R.string.
        structure_add_dialog_title);
        //builder2.setIcon(R.drawable.icon);

        final EditText Edit_structureNameNew = (EditText) dialogView.findViewById(R.id.
        editStructureName);
        final EditText Edit_structureDescNew = (EditText) dialogView.findViewById(R.id.
        editStructureDescription);

        builder.setPositiveButton(R.string.save,
        new DialogInterface.OnClickListener() {

        });

    }
public void onClick(DialogInterface dialog, int which) {

    // Defines strings input in by
    // the user
    String structureName =
        Edit_structureNameNew.getText().toString();
    String structureDesc =
        Edit_structureDescNew.getText().toString();

    // Adds the user-defined structure
    // to the database
    utilsHelper.dbHelper.
        createStructureEntry(
            structureName, structureDesc);

    // Refreshes the list view screen
    // with new changes
    updateList();

    return;
}

builder.setNegativeButton(android.R.string.cancel,
    new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int which) {
            return;
        }
    });

Dialog structureDialog = builder.create();
return structureDialog;

case STRUCTURE_EDIT_DIALOG_ID:
    AlertDialog.Builder builder2 = new AlertDialog.Builder(this);

    // Inflates the custom xml file layout
    create_structure_dialog.xml
LayoutInflater inflater2 = this.getLayoutInflater();
View dialogView2 = inflater2.inflate(R.layout.
create_structure_dialog, null);
bUILDER2.setView(dialogView2);

builder2.setTitle(R.string.
structure_edit_dialog_title);
//builder2.setIcon(R.drawable.icon);

final EditText Edit_structureNameEdit = (EditText)
dialogView2.findViewById(R.id.
editStructureName);
final EditText Edit_structureDescEdit = (
EditText) dialogView2.findViewById(R.id.
editStructureDescription);

builder2.setPositiveButton(R.string.save,

new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface
dialog, int which) {

        // Defines strings input in by
        the user
        String structureName =
            Edit_structureNameEdit.getText
            ().toString();
        String structureDesc =
            Edit_structureDescEdit.getText
            ().toString();

        // Updates the user-defined
        structure to the database
        //structureID = adapter.getCursor().
        getInt(adapter.getCursor().
        getColumnIndex(SqlHelper.COLUMN_ID)
        );
        String[][] databaseUpdates = {
            {SqlHelper.COLUMN_STRUCTURE_NAME,
                structureName},
            {SqlHelper.COLUMN_STRUCTURE_DESC,
                structureDesc}
        };
        utilsHelper.dbHelper.
        updateStructureEntry(
databaseUpdates, structureID );
/ Refersehs the list view screen with new changes
updateList();

return;
}
});

builder2.setNegativeButton(android.R.string.cancel,
    new DialogInterface.OnClickListener()
    {public void onClick(DialogInterface dialog, int which)
    {return;
    }});

Dialog structureDialogEdit = builder2.create();
return structureDialogEdit;

return null;

}@Override
protected void onPrepareDialog(int id, Dialog dialog) {

switch (id) {
    case STRUCTURE_ADD_DIALOG_ID:
        // Clear the input box.
        final EditText structureNameNew = (EditText) dialog.findViewById(R.id.editStructureName);
        final EditText structureDescNew = (EditText) dialog.findViewById(R.id.editStructureDescription);

        // Clear the EditText fields
        structureNameNew.setText("");
        structureDescNew.setText("");
        break;

    case STRUCTURE_EDIT_DIALOG_ID:

100
final EditText structureNameEdit = (EditText) dialog.findViewById(R.id.editStructureName);
final EditText structureDescEdit = (EditText) dialog.findViewById(R.id.editStructureDescription);

// Clear the EditText fields
String oldName = this.currentCursor.getString(this.currentCursor.getColumnIndex(SqlHelper.COLUMN_STRUCTURE_NAME));
String oldDesc = this.currentCursor.getString(this.currentCursor.getColumnIndex(SqlHelper.COLUMN_STRUCTURE_DESC));

structureNameEdit.setText(oldName);
structureNameEdit.requestFocus();
structureDescEdit.setText(oldDesc);
break;

default:
    Log.d(LAYOUT_INFLATER_SERVICE, "Wrong Structure DIALOG ID");
    break;
}

@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.structure_list_menu, menu);
}

@Override
public boolean onContextItemSelected(MenuItem item) {
    AdapterContextMenuInfo info = (AdapterContextMenuInfo) item.getMenuInfo();
    structureID = utilsHelper.dbHelper.getStructureID(currentCursor);
}

switch (item.getItemId()) {
    case R.id.openStructure:
        utilsHelper.startNextActivity(
            structureID, questionnaire_Activity1.class);
        return true;
    case R.id.editStructure:
        showDialog(STRUCTURE_EDIT_DIALOG_ID);
        return true;
    case R.id.deleteStructure:
        // TODO: Create a "are you sure, prompt "
        utilsHelper.dbHelper.delete_byID(
            structureID);
        updateList();
        return true;
    case R.id.uploadStructure:
        httpPostHelper httpPost = new
            httpPostHelper(mContext);
        httpPost.postToRemoteDatabase(
            structureID);
        return true;
}
    return false;
}

private void updateList(){
    // Refreshes the list view to incorporate the newly
    added/deleted structure
    currentCursor.requery();
    adapter.notifyDataSetChanged();
}

@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    if ((keyCode == KeyEvent.KEYCODE_BACK)) {
        utilsHelper.closeDatabase();
        Intent startMain = new Intent(Intent.ACTION_MAIN);
        startMain.putExtra(Intent.EXTRA_USER);
D.2 Questionnaire Activities

D.2.1 Screen 1

```java
package com.example.haiti.infrastructure.surveyor;

import android.app.Activity;
import android.app.ListActivity;
import android.content.Context;
import android.content.Intent;
import android.database.Cursor;
import android.database.SQLException;
import android.os.Bundle;
import android.view.KeyEvent;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageButton;
import android.widget.TextView;
import android.widget.Toast;

public class questionnaire_ACTIVITY1 extends Activity {

    private Utils utilsHelper = null;
    private Cursor structureCursor = null;
    private int structureID;
    public Context mContext;
    private EditText Edit_surveyorName = null;
    private EditText Edit_surveyorPhoneNumber = null;
    private EditText Edit_structureAddress = null;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_questionnaire);

        // Initialize UI elements
        Edit_surveyorName = findViewById(R.id.edit_surveyorName);
        Edit_surveyorPhoneNumber = findViewById(R.id.edit_surveyorPhoneNumber);
        Edit_structureAddress = findViewById(R.id.edit_structureAddress);

        // Set listeners
        Edit_surveyorName.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // Handle surveyor name input
            }
        });

        // Other UI element listeners...
    }

    // Main activity methods...
}
```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.questionnaire_layout_1);

    utilsHelper = new Utils(questionnaire_Activity1.this);
    utilsHelper.openDatabase();
    getStructure();
    manageEditTextFields();
    setupNextButton();
} // end of the onCreate method

@Override
protected void onPause() {
    super.onPause();
    utilsHelper.closeDatabase();
}

@Override
protected void onStop() {
    super.onStop();
    utilsHelper.closeDatabase();
}

private void getStructure(){
    // Get structure ID
    structureID = getIntent().getIntExtra("structureID", 1000); // Get structureID otherwise 1000
    structureCursor = utilsHelper.getStructureCursor(structureID);
}

private void manageEditTextFields(){
    // Declares the text input fields
    Edit_surveyorName = (EditText) this.findViewById(R.id.editFullName);
    Edit_surveyorPhoneNumber = (EditText) this.findViewById(R.id.editPhoneNumber);
    Edit_structureAddress = (EditText) this.findViewById(R.id.editAddress);
    // Checks if data already exists
String oldFullName = structureCursor.getString(
structureCursor.getColumnIndex(SqlHelper.
COLUMN_SURVEYOR_NAME));
String oldPhoneNumber = structureCursor.
getString(structureCursor.getColumnIndex(
SqlHelper.COLUMN_SURVEYOR_PHONE_NUMBER));
String oldAddress = structureCursor.getString(
structureCursor.getColumnIndex(SqlHelper.
COLUMN_STRUCTURE_ADDRESS));

// Sets the text of the input fields
Edit_surveyorName.setText(oldFullName);
Edit_surveyorPhoneNumber.setText(
oldPhoneNumber);
Edit_structureAddress.setText(oldAddress);

private void setupNextButton(){

    // Add Next button
    Button btnNextQuestions = (Button) findViewById(R.id.
        buttonNext);
    btnNextQuestions.setOnClickListener(new
    OnClickListener() {

        public void onClick(View v) {

            // Defines strings input in by the user
            String surveyorName = Edit_surveyorName.
                getText().toString();
            String surveyorPhoneNumber =
            Edit_surveyorPhoneNumber.getText().toString
            ();
            String structureAddress =
            Edit_structureAddress.getText().toString();

            String[][] databaseUpdates = {
            {SqlHelper.
                COLUMN_SURVEYOR_NAME, 
                surveyorName},
            {SqlHelper.COLUMN_STRUCTURE_ADDRESS, 
                structureAddress},
            {SqlHelper.
                COLUMN_SURVEYOR_PHONE_NUMBER, 
                surveyorPhoneNumber},

            // More updates...
{SqlHelper.COLUMN_STRUCTURE_STATUS, "In Progress"};

};
utilsHelper.dbHelper.updateStructureEntry(
databaseUpdates, structureID);

utilsHelper.startNextActivity(structureID,
questionnaire_Activity2.class);

}
}); // End of Next button click listener

// Back Button Pressed
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
  if ((keyCode == KeyEvent.KEYCODE_BACK)) {
    utilsHelper.startNextActivity(structureID,
main_Activity.class);
  }
  return super.onKeyDown(keyCode, event);
}

D.3 Exterior Activity

package com.example.haiti.infrastructure.surveyor;

import java.io.File;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.os.Environment;
import android.provider.MediaStore;
import android.util.Log;
import android.view.KeyEvent;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ImageButton;
import android.widget.EditText;
import android.widget.Spinner;
import android.widget.Toast;

public class exterior_Activity extends Activity {

    private Utils utilsHelper = null;
    private Cursor structureCursor = null;
    private int structureID;
    private Context mContext = null;
    private static int CAPTURE_IMAGE = -1;
    private static final int CAPTURE_IMAGE_FRONT = 0;
    private static final int CAPTURE_IMAGE_RIGHT = 1;
    private static final int CAPTURE_IMAGE_BACK = 2;
    private static final int CAPTURE_IMAGE_LEFT = 3;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.exterior_layout);
        mContext = this;
        utilsHelper = new Utils(exterior_Activity.this);
        utilsHelper.openDatabase();

        // Initializes view
        getStructure();
        setupPhotoButtons();
        setupNextButton();
    } // end of the onCreate method

    @Override
    protected void onPause() {
        super.onPause();
        utilsHelper.closeDatabase();
    }

    @Override
    protected void onStop() {
        super.onStop();
        utilsHelper.closeDatabase();
    }
}
private void getStructure() {
    // Get structure ID
    structureID = getIntent().getIntExtra("structureID", 1000); // Get structureID otherwise 1000
    structureCursor = utilsHelper.getStructureCursor(structureID);
}

private void setupPhotoButtons() {

    // Front Photo Button
    ImageButton btnFront = (ImageButton) findViewById(R.id.btnExteriorFront);

    if (photoAlreadyTaken(CAPTURE_IMAGE_FRONT)) {
        btnFront.setImageResource(R.drawable.check);
    }
    btnFront.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
            takePhoto(CAPTURE_IMAGE_FRONT);
        }
    }); // End of Next button click listener

    // Right Photo Button
    ImageButton btnRight = (ImageButton) findViewById(R.id.btnExteriorRight);

    if (photoAlreadyTaken(CAPTURE_IMAGE_RIGHT)) {
        btnRight.setImageResource(R.drawable.check);
    }
    btnRight.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
            takePhoto(CAPTURE_IMAGE_RIGHT);
        }
    }); // End of Next button click listener
// Back Photo Button
ImageButton btnBack = (ImageButton) findViewById(R.id.
btnExteriorBack);

if (photoAlreadyTaken(CAPTURE_IMAGE_BACK)) {
    btnBack.setImageResource(R.drawable.check);
}

btnBack.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
        takePhoto(CAPTURE_IMAGE_BACK);
    }
}); // End of Next button click listener

// Left Photo Button
ImageButton btnLeft = (ImageButton) findViewById(R.id.
btnExteriorLeft);

if (photoAlreadyTaken(CAPTURE_IMAGE_LEFT)) {
    btnLeft.setImageResource(R.drawable.check);
}

btnLeft.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
        takePhoto(CAPTURE_IMAGE_LEFT);
    }
}); // End of Next button click listener

private boolean photoAlreadyTaken(int photoSide) {
    boolean exists = false;
    String photo_value = null;

    switch (photoSide) {
        case CAPTURE_IMAGE_FRONT:
            photo_value = structureCursor.getString(
                structureCursor.getColumnIndex(SqlHelper.
COLUMN_STRUCTURE_PHOTO_FRONT));
            break;
    }
    return exists;
}
case CAPTURE_IMAGE_RIGHT:
    photo_value = structureCursor.getString(
        structureCursor.getColumnIndex(SqlHelper.
            COLUMN_STRUCTURE_PHOTO_RIGHT));
    break;

case CAPTURE_IMAGE_BACK:
    photo_value = structureCursor.getString(
        structureCursor.getColumnIndex(SqlHelper.
            COLUMN_STRUCTURE_PHOTO_BACK));
    break;

case CAPTURE_IMAGE_LEFT:
    photo_value = structureCursor.getString(
        structureCursor.getColumnIndex(SqlHelper.
            COLUMN_STRUCTURE_PHOTO_LEFT));
    break;

default:
    Log.d(LAYOUT_INFLATER_SERVICE, "IMAGE EXISTS ERROR");
    break;
}

if (photo_value != null){
    exists = true;
}

return exists;

private void takePhoto(int photoSide){

    // Determines which photo
    CAPTURE_IMAGE = photoSide;

    String photoPath = getPhotoPath(photoSide);

    File outFile = new File(photoPath);
    Uri iImageOutputUri = Uri.fromFile(outFile);

    // Start camera intent to capture image and store the file locally
    // REQ: 3.4.4.R3 met
    Intent intent = new Intent(MediaStore.
        ACTION_IMAGE_CAPTURE);


intent.putExtra(MediaStore.EXTRA_OUTPUT, iImageOutputUri);

utilsHelper.closeDatabase();
(Activity)mContext).startActivityForResult(intent, CAPTURE_IMAGE);
}

private String getPhotoPath(int photoSide) {

    File file = Environment.getExternalStorageDirectory();
    String structureName = structureCursor.getString(
        structureCursor.getColumnIndex(SqlHelper.COLUMN_STRUCTURE_NAME));
    String photo_directory = file.getAbsolutePath() + "/Haiti_Structure_Photos/" + structureName;

    // Creates new directory if needed
    File dir = new File(photo_directory);
    if (!dir.isDirectory())
        dir.mkdirs();

    String photoSideText = "";

    switch (photoSide) {
        case CAPTURE_IMAGE_FRONT:
            photoSideText = "Front";
            break;
        case CAPTURE_IMAGE_RIGHT:
            photoSideText = "RIGHT";
            break;
        case CAPTURE_IMAGE_BACK:
            photoSideText = "Back";
            break;
        case CAPTURE_IMAGE_LEFT:
            photoSideText = "Left";
            break;
        default:
            Log.d(LAYOUT_INFLATER_SERVICE, "Wrong Image");
            break;
    }
}
String photoPath = photo_directory + File.separator + structureName + "_" + photoSideText + ".jpg";

return photoPath;
}

protected void onActivityResult(int requestCode, int resultCode, Intent data) {

if (resultCode == RESULT_OK) {

utilsHelper.openDatabase();

if (requestCode == CAPTURE_IMAGE_FRONT) {
    String photoPath = getPhotoPath(CAPTURE_IMAGE_FRONT);
    File photo = new File(photoPath);

    if (photo.exists()) {

    String[][] databaseUpdates = {
        {SqlHelper.COLUMN_STRUCTURE_PHOTO_FRONT, photoPath},
    };
    
    utilsHelper.dbHelper.
        updateStructureEntry(databaseUpdates, structureID);

    ImageButton btnFront = (ImageButton)
        findViewById(R.id.btnExteriorFront);
    btnFront.setImageResource(R.drawable.check);
    Toast.makeText(mContext, "Photo saved", Toast.LENGTH_SHORT).show();
    
    }

}

}
if (requestCode == CAPTURE_IMAGE_RIGHT)
{
    String photoPath = getPhotoPath(CAPTURE_IMAGE_RIGHT);
    File photo = new File(photoPath);

    if (photo.exists()){
        String[][] databaseUpdates = {
            {SqlHelper.COLUMN_STRUCTURE_PHOTO_RIGHT, photoPath},
        };

        utilsHelper.dbHelper.
            updateStructureEntry(
                databaseUpdates, structureID);

        ImageButton btnRight = (ImageButton) findViewById(R.id.btnExteriorRight);
        btnRight.setImageResource(R.drawable.check);
        Toast.makeText(mContext, "Photo saved", Toast.LENGTH_SHORT).show();
    }
}

if (requestCode == CAPTURE_IMAGE_BACK)
{
    String photoPath = getPhotoPath(CAPTURE_IMAGE_BACK);
    File photo = new File(photoPath);

    if (photo.exists()){
        String[][] databaseUpdates = {
            {SqlHelper.COLUMN_STRUCTURE_PHOTO_BACK, photoPath},
        };

        utilsHelper.dbHelper.
            updateStructureEntry(
                databaseUpdates, structureID);
    }
ImageButton btnBack = (ImageButton) findViewById(R.id.btnExteriorBack);
btnBack.setImageResource(R.drawable.check);
Toast.makeText(mContext,"Photo saved",Toast.LENGTH_SHORT).show();

if (requestCode == CAPTURE_IMAGE_LEFT)
{
    String photoPath = getPhotoPath(CAPTURE_IMAGE_LEFT);
    File photo = new File(photoPath);
    if (photo.exists()){

        String[][] databaseUpdates = {
        {SqlHelper.COLUMN_STRUCTURE_PHOTO_LEFT, photoPath},
        };

        utilshelper.dbHelper.updateStructureEntry(databaseUpdates, structureID);

        ImageButton btnLeft = (ImageButton) findViewById(R.id.btnExteriorLeft);
btnLeft.setImageResource(R.drawable.check);
Toast.makeText(mContext,"Photo saved",Toast.LENGTH_SHORT).show();

    }
} // IF RESULT == OK

} // On Activity Result

114
private void setupNextButton() {

    // Add Next button
    Button btnNextQuestions = (Button) findViewById(R.id.buttonNext);
    btnNextQuestions.setOnClickListener(new OnClickListener() {

        public void onClick(View v) {

            utilsHelper.startNextActivity(structureID, main_Activity.class);
        }
    }); // End of Next button click listener
}

// Back Button Pressed
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    if ((keyCode == KeyEvent.KEYCODE_BACK)) {
        utilsHelper.startNextActivity(structureID, questionnaire_Activity3.class);
    }
    return super.onKeyDown(keyCode, event);
}
}
APPENDIX E

INFRASTRUCTURE AS CODE PUPPET MODULES

Most puppet modules consisted of minor to significant changes from the open source version available. The puppet modules for Celery, Java (specific version), Lettuce, and RabbitMQ were not publicly available during the time of creation for the CentOS operating system so they were created manually.

E.1 Celery

class celery()
{
    # installs package needed to install celery
    package { "python-setuptools":
        ensure => "installed"
    }

    # installs system libraries
    package { "gsl":
        ensure => "installed"
    }

    # installs postgrsot support for celery tasks
    package { "postgresql-devel":
        ensure => "installed"
    }

    # installs postgres support for celery tasks
    package { "python-psycopg2":
        ensure => "installed"
    }

    # Installs celery globally
}
exec { "easy_install Celery":
  require => Package['python-setuptools'],
  path => "'/usr/local/bin:/usr/bin:/bin",
  unless => "which celery",
}

# Copy celeryd-initd
file {'/etc/init.d/celeryd':
  ensure => "file",
  source => "puppet:///modules/celery/celeryd-initd",
  alias => "celeryd-init",
}

# Create /etc/default directory
file {'/etc/default':
  ensure => "directory",
}

# Create /var/log/celery
file {'/var/log/celery':
  ensure => "directory",
}

# Copy celeryd-config
file {'/etc/default/celeryd':
  ensure => "file",
  source => "puppet:///modules/celery/celeryd-config",
}

# Create /omdata and copy
file {'/omdata':
  ensure => "directory",
  recurse => "true",
  source => "puppet:///modules/celery/omdata",
}

exec {"chkconfig --level 345 celeryd on":
  command => "'/sbin/chkconfig --level 345 celeryd on",
  require => File['celeryd-init'],
}
E.2 Java

This module was adapted from a version that was included on the PuppetLabs website.

E.2.1 init.pp file

```puppet
# /etc/puppet/modules/java/manifests/init.pp
class java {

  require java::params

  file {
    $java::params::java_base:
      ensure        => "directory",
      owner         => "root",
      group         => "root",
      alias         => "java-base",
  }

  file {
    $java::params::java_base/jdk$java::params::java_version.tar.gz:
      ensure      => file,
      mode        => 0644,
      owner       => root,
      group       => root,
      source      => "puppet:///modules/java/
                jdk$java::params::java_version.tar.gz",
      alias       => "java-source-tgz",
      before      => Exec["untar-java"],
      require     => File["java-base"],
  }

  exec {
    untar $java::params::java_base/jdk$java::params::java_version.tar.gz:
      command => "/bin/tar -zxf $java::params::java_base/jdk$java::params::java_version.tar.gz",
      cwd     => "$java::params::java_base",
      creates => "$java::params::java_base/jdk$java::params::java_version",
      alias   => "untar-java",
      refreshonly => true,
  }
}
```
subscribe => File["java-source-tgz"],
before => File["java-app-dir"],
}

file { "${java::params::java_base}/jdk${java::params::
java_version}"
ensure => "directory",
mode => 0644,
owner => root,
group => root,
alias => "java-app-dir",
}

exec { "Set default javac"
command => "/usr/sbin/update-
alternatives --install /usr/bin/javac javac
/opt/java/jdk${java::params::java_version} /
bin/javac 1",
alias => "config_javac",
refreshonly => true,
subscribe => File["java-app-dir"],
}

exec { "Set default java"
command => "/usr/sbin/update-
alternatives --install /usr/bin/java java /
/opt/java/jdk${java::params::java_version} /
bin/java 1",
alias => "config_java",
refreshonly => true,
subscribe => File["java-app-dir"],
}

exec { "Set default javaws"
command => "/usr/sbin/update-
alternatives --install /usr/bin/javaws
javaws /opt/java/jdk${java::params::
java_version} /bin/javaws 1",
alias => "config_javaws",
refreshonly => true,
subscribe => File["java-app-dir"],
}

exec { "Set default javadoc"
command => "/usr/sbin/update-
alternatives --install /usr/bin/javadoc
javadoc /opt/java/jdk${java::params::
java_version} /bin/javadoc 1",

java_version }/bin/javadoc 1",
alias => "config_javadoc",
refreshonly => true,
subscribe => File["java-app-dir"],
}

exec { "Set default javah":
command => "/usr/sbin/update-alternatives --install /usr/bin/javah javah
/opt/java/jdk${java::params::java_version}/bin/javah 1",
alias => "config_javah",
refreshonly => true,
subscribe => File["java-app-dir"],
}

exec { "Set default javap":
command => "/usr/sbin/update-alternatives --install /usr/bin/javap javap
/opt/java/jdk${java::params::java_version}/bin/javap 1",
alias => "config_javap",
refreshonly => true,
subscribe => File["java-app-dir"],
}

}

E.2.2 params.pp file

# /etc/puppet/modules/java/manifests/params.pp
class java::params {
    $java_version = $::hostname ? {
        default => "1.6.0_37",
    }

    $java_base = $::hostname ? {
        default => "/opt/java/",
    }
}

E.3 Lettuce
class lettuce()
{
    package { "python-setuptools":
        ensure => "installed"
    }
    exec { "easy_install lettuce":
        require => Package['python-setuptools'],
        path => "/usr/local/bin:/usr/bin:/bin",
        unless => "which lettuce",
    }
}

E.4 RabbitMQ

# RabbitMQ Server puppet file

class rabbit-mq {
    $key = "http://www.rabbitmq.com/rabbitmq-signing-key-
           public.asc",
    $version="3.0.2",
    $release_version="1",
}

# Installs epel for 64 bit
exec { "install-epel":
    command => "/bin/rpm -i http://dl.fedoraproject.org/
             pub/epel/6/x86_64/epel-release-6-8.noarch.rpm",
}

# Installs erlang package
package { 'erlang':
    ensure => installed,
    provider => yum,
    require => Exec["install-epel"],
}

exec { "rpm --import ${key}":
    path => ["/bin","/usr/bin","/sbin","/usr/sbin"],
}

# installs rabbitmq server
package { "rabbitmq-server":
    provider => rpm,
    ensure => installed,
    source => "http://www.rabbitmq.com/releases/rabbitmq-
              server/v${version}/rabbitmq-server-${version}-${

release_version}.noarch.rpm",
    require => Exec["rpm --import ${key}"],
  }

# Starts the rabbitmq server process
service { 'rabbitmq-server':
  ensure => running,
  enable => true,
  hasrestart => true,
}

# Opens firewall port
firewall { '999 allow port 5672':
  action => accept,
  dport => '5672',
  proto => 'tcp',
}

}
BIBLIOGRAPHY

1. 10+ deploys per day: Dev and ops cooperation. URL http://www.slideshare.net/jallspaw/10-deploys-per-day-dev-and-ops-cooperationat-flickr.
16. Agile system administration e-mail list. URL groups.google.com/group/agile-system-administration.


26. Engineering to Empower. URL http://e2e.nd.edu/.


33. Who is Using Puppet?


