A Web Based Framework for Pre-release Testing of Mobile Applications

Abeer Hamdy[1,2,a], Osman Ibrahim[1], Ahmed Hazem[1]

[1] *Informatics and Computer science Faculty, The British University in Egypt, Cairo, Egypt*
[2] *Computers and Systems department, Electronics Research Institute, Cairo, Egypt*

Abstract. Mobile applications are becoming an integral part of daily life and of business's marketing plan. They are helpful in promoting for the business, attracting and retaining customers. Software testing is vital to ensure the delivery of high quality mobile applications that could be accessed across different platforms and meet business and technical requirements. This paper proposes a web based tool, namely Pons, for the distribution of pre-release mobile applications for the purpose of manual testing. Pons facilities building, running, and manually testing Android applications directly in the browser. It gets the developers and end users engaged in testing the applications in one place, alleviates the tester's burden of installing and maintaining testing environments, and provides a platform for developers to rapidly iterate on the software and integrate changes over time. Thus, it speeds up the pre-release testing process, reduces its cost and increases customer satisfaction.

1 Introduction

Mobile platforms have been getting powerful and sophisticated which facilitates the development of a vast number of mobile applications that serve the business world. It became a business practice for small and large corporations to have a mobile application. Corporations use mobile applications for several purposes including promotion for and selling their products, better communication and engagement with customers. Examples of those applications include Uber [1], AEMC [2] and Empire loan [3]. Uber is a service that offers users safe and reliable transportation. Uber has a network of tens of thousands (maybe even hundreds of thousands) of drivers who respond to user requests for rides. While, Empire Loan has a chain of pawn shops that people can buy and sell merchandise in. AEMC is a worldwide energy monitoring company manufactures hardware that monitors energy consumption in large buildings and facilities.

In this extremely competitive business world, companies seek to release applications that are smart, simple and function well on different mobile devices with different hardware configurations, screen sizes and resolutions, different versions of operating systems, and across different mobile networks. Various testing methodologies were suggested for testing the different aspects of mobile applications before they are released which include usability testing [4],[5], GUI testing [6], [7], performance testing, compatibility testing, privacy testing [8]. Testing could be performed manually or automatically [6], [7]. Automatic testing is useful for big scenarios which are time consuming when run manually or for regression testing. While manual testing allows testers to perform ad-hoc tests, it is important in the pre-release stage. However, the existing platforms for pre-release distribution of mobile applications for the purpose of manual testing are badly-designed and often lead to various problems [8] such as:

1. Download and installation takes up a lot of time: To start testing an application, it must be downloaded and installed on a mobile device. Such process delays the development progress.
2. Feedback is cluttered among different mediums: In order to gather feedback, developers usually have to arrange for a way through which testers can leave their comments. Most of testing tools does not offer a medium for developer and testers to collaborate and communicate, leaving them to traditional communication practices (such as email communications). This leads to cluttered feedback, resulting in confusion for the developer and longer times in processing issues, feature requests, and bug reports.
3.Testing is done in isolation and out of context: Testing of mobile applications is often isolated and is done out of context. That is either due to testers having access to the software but not to the developer while carrying out their task, or by developers themselves trying to imitate the tester behavior without taking into consideration their perspectives are almost always quite different.
4. No over-the-air updates: Existing solutions does not offer developers a way to update the software over-the-air. Essentially, this means more time is spent building a new version of the application to be distributed, which can be a huge amount of time depending on the complexity of the components used and the size of the application itself.

[a] Corresponding author: abeer.hamdy@bue.edu.eg

This paper proposes a web based tool, namely Pons, which:

1. Provides a platform for distributing the pre-release of mobile applications for the purpose of manual testing.
2. Facilitates building, running and manually testing mobile applications in a browser.
3. Manages the process of manual testing for the purpose of speeding up the testing process and reducing its effort and cost.
4. Allows the testers to focus on testing mobile applications without having to worry about getting the platform up and running or the environment configured.
5. Provides a medium for both of developers and testers to collaborate and communicate without isolation from the application they are working on, Consequently, speeding up the testing process.

The paper is organized as follows: Section 2 provides the structure of Pons, while, section 3 discusses its development. Section 4 concludes the paper and discusses how Pons can be extended in the future.

2 Pons architecture

Pons is a web-based tool that facilitates pre-release distribution and engage the mobile application developers and clients /testers in a constructive closed feedback loop to achieve the business objectives of the applications. It comprises four main components, as demonstrated by figure 1, which are: 1) A web application, 2) Containers which work as a sandbox environment to build the mobile applications [9], 3) Version control management component to keep track of the different versions of the mobile application [10], and 4) Live streaming component to stream the mobile application to the client [11]. The roles of these components and their development are discussed in the following section.

3 Pons development and technologies

The following subsections discuss the technologies used in developing the different components of Pons and the role of each of them.

3.1 Web application component

Pons's web application is implemented as a single page application (SPA) which comprises two separate applications, front-end and back-end, that communicate using a JSON-based API [14], as depicted by figure 2. The front-end is built using a JavaScript web application framework Ember.JS [12]. While, the back-end is built using the Ruby on Rail framework [13].

For the past couple of years, there has been a growing trend in web development that promotes moving the application logic and user interface rendering (among other things) to the client. These applications have come to be known as single page applications (SPA). Figure 3 outlines the architecture of SPAs. In such an architecture the back-end application is responsible for the data layer (manipulating and storing data, along with exposing a REST API that is consumed by clients), while the front-end application is responsible for application-logic and the presentation layers (view rendering, and handling requests, among other tasks).

As Pons web application is developed as a SPA, its back-end is different from typical Ruby on Rail application in removing the entire View layer (since no template generation or view rendering is to be done on the server-side). This means Ruby on Rail application would serve as a back-end, providing persistence and data management through a REST API. Figure 4 outlines the architecture of Pons's Ruby on Rail application. As shown in the figure, there is no "Action View" module (as in typical Ruby on Rail) while there is a new module called ActiveModel::Serializers. This new module is responsible for serializing and deserializing JSON requests and responses. The rationale behind moving view rendering away from the server-side in Pons can be narrowed down to the following two reasons: i. Performance: Instead of having to send a full-blown HTTP request over the network each time a user interacts with the application, the entire payload is loaded upfront (hence the name, single page refers to that payload which is loaded on the first request only), allowing for communications with the server to be minimal, that is, to be only for fetching data. This has the advantage of keeping the user interface snappy and responsive. The overall performance of the application increases consequently. ii. Coherence: Rendering on the server side adds a layer of complexity, arising from the fact developers by nature already do some rendering on the client (most notably, in the form of UI interactivity, such as sliders, widgets, and rich text editors). This actually means having state and behaviour of UI duplicated, both on the client and the server, implemented using two different programming languages and running on two different machines. Rendering on the client eliminates this problem, and keeps the UI logic consistent and coherent.

This decision does not mean that client-side rendering is bullet-proof. As with any other kind of technical choice, opting for client-side rendering has its own disadvantages and technical debt. However, this project assumes that its benefits outweigh its problems.

Pons's front end is developed using Ember.js. Ember.js is based on the Model-View-Controller (MVC) paradigm. Ember facilitates creating complex front-end

applications. Figure 5 outlines the interaction between different components in an Ember.js application, while Fig.6 describes how a request is routed and handled by Emper components.

3.2 Version management component

Generally speaking, version (Release) Management is the practice of keeping track of changes in a set of files, in order to retrieve them later on. Tools for controlling software versions are called Version Control Management Systems (VCMS). These systems provide a number of important functions that are vital to any software development activity [4] such as assigning a unique identifier to each version, storing and listing all changes to the code and providing a storage management to ensure code is not stored multiple times as new versions are recorded. VCMS also allow a single file to be edited concurrently by more than one developer.
Pons adopts a distributed version control management system called GIT [10]. GIT organizes the code into repositories, which are basically folders organized in a specific structure or format. Aside from that physical structure, a repository is also structured in a logical way, dividing the code inside to branches, tags and commits. A branch is a deviation from the original codeine, giving a developer the chance to write experimental code without worrying about the original version of his/her work. This has the advantage of allowing context switching without having a huge impact on the system: one can create a branch, do some work, go back to original branch, fix something, then return to the new branch and merge his/her work.
Mobile application developer should have a GITHub account to be able to use Pons in developing an application. Any client, to start accessing an existing mobile application, should request a permission through Pons.

3.3 Containers

Pons utilizes a container technology called Docker [14], [15], [16]. Docker is a portable container that wraps up a piece of software in a complete file system that contains everything it needs to run: code, runtime, system tools, and system libraries. Container is a lightweight alternative to a full-fledged virtual machine such as KVM [17]. Containers isolate applications from each other and the underlying infrastructure while providing an added layer of protection for the application. This guarantees that it will always run the same, regardless of the environment it is running in. Virtual machines run their own kernel and operating system instances and provide good isolation but that comes with the cost of having overhead. While containers are less isolated but have much lower overhead as both of the kernel and operating system instances are shared.
Docker is an extension to the capabilities of Linux containers (LXCs) [18], it wraps Linux containers and provide better abstraction and tooling. It was started as a side project and only open-sourced in 2013. It should be mentioned that Docker containers are based on open standards allowing containers to run on all major Linux distributions and Microsoft operating systems with support for every infrastructure. Pons utilizes Docker to create an LXC based container each time an Android application is built; to sandbox, the application and streamline the whole build/running process as depicted by figure 7.

Pons utilizes Docker by predefining Docker images that contain the necessary services and tools to build android applications, starting from the operating system up to the software development kit (SDK). A container is then built using one of these images to store the source code of a mobile application at specific moment of history in a sandbox environment.
Ready-made Docker images (snapshots of a container) are stored on a highly scalable server application called Docker registry [16]. There are two types of registries for Docker, public and private. Pons uses the private Docker registry to keep Docker images on the same server and consequently pulling them would be much faster, and obviously, to keep them private since Dockers public registry can be accessed by anyone. Docker pulls and pushes (stores and retrieves) images and repositories from/to the private Docker registry installed on the server. An alternative would be to pre-build these images and store (pull) them on the local Docker instance, which has the same effect as keeping them in a private registry in terms of speed and security.
The images stored are built in a down-top approach as shown in figure 8, meaning that each one would be depending on the other from the top down to the base image which will only include the operating system (in our case, Ubuntu). Such setup would allow for better configuration, as one can only include what is needed for a specific build.

Docker provides an application-programming interface (API) [9] to control a Docker instance remotely. This API is used by Pons to control the Docker daemon, to create, build and delete images and containers.

3.4 Open SSH

Due to a Docker limitation that prevents a Linux container from running more than one process, OpenSSH is employed by Pons. OpenSSH [19] is a collection of network-level utilities aimed at facilitating secure connections between different machines. The rationale behind installing an OpenSSH server in each container is to allow running different commands and executing multiple processes inside the container such as: Cloning a GIT repository, Creating an Android emulator.
Issuing commands to the container is done through Rye [20], which is a Ruby library that allow running SSH commands on multiple machines. This setup works as outlined in figure 9.

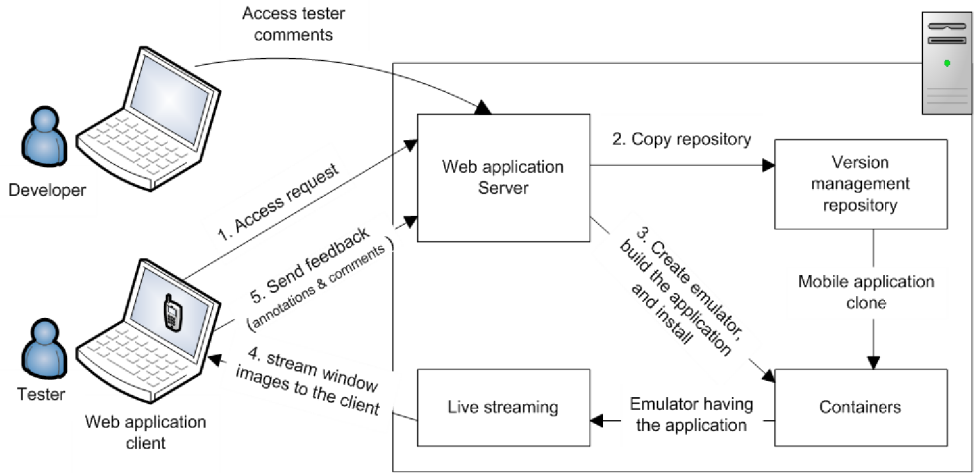

Figure1. Pons Architecture

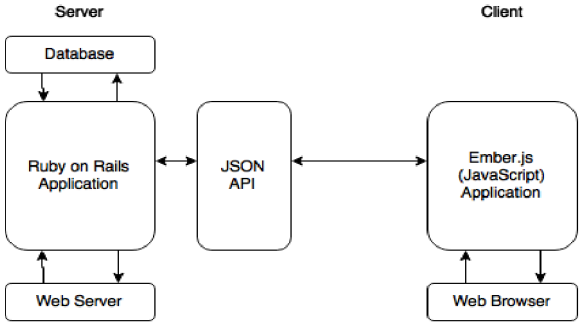

Figure 2. Pons web application architecture.

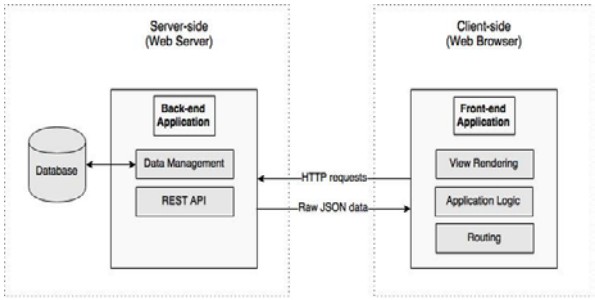

Figure 3. Single Page application (SPA) architecture.

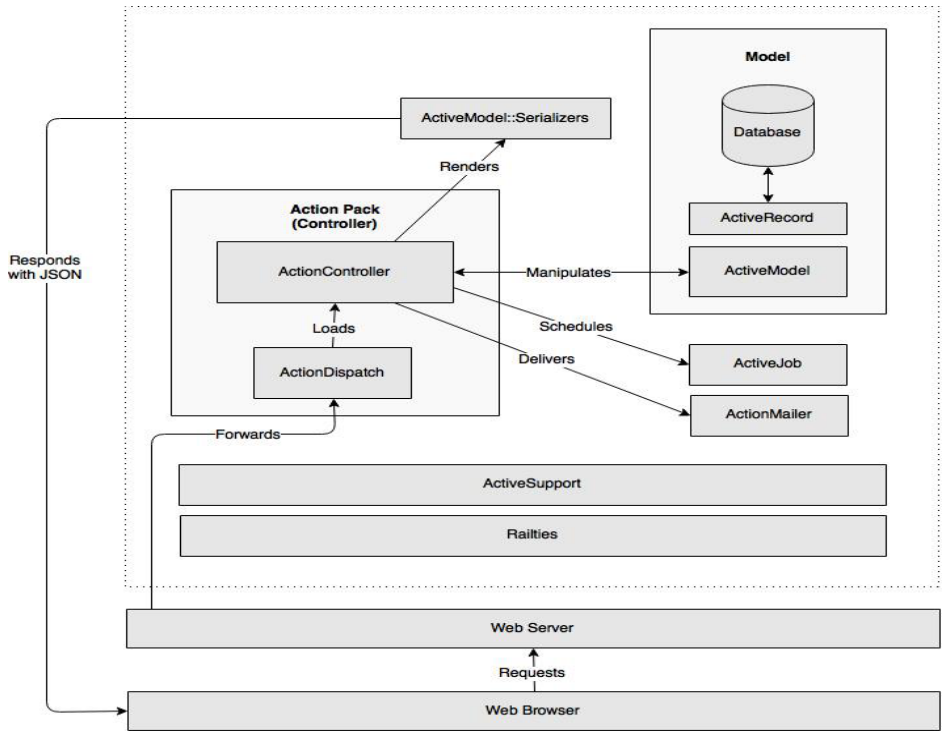

Figure 4. Pons's Ruby on Rails application

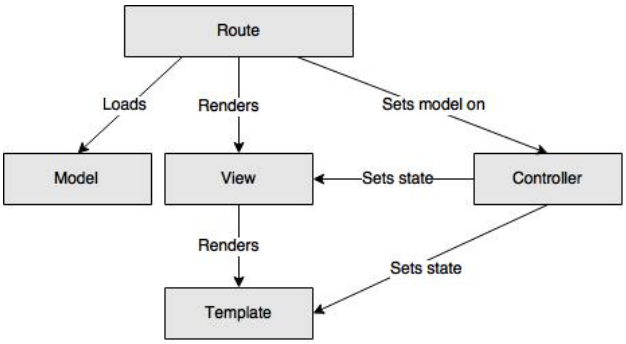

Figure 5. Ember.js application architecture

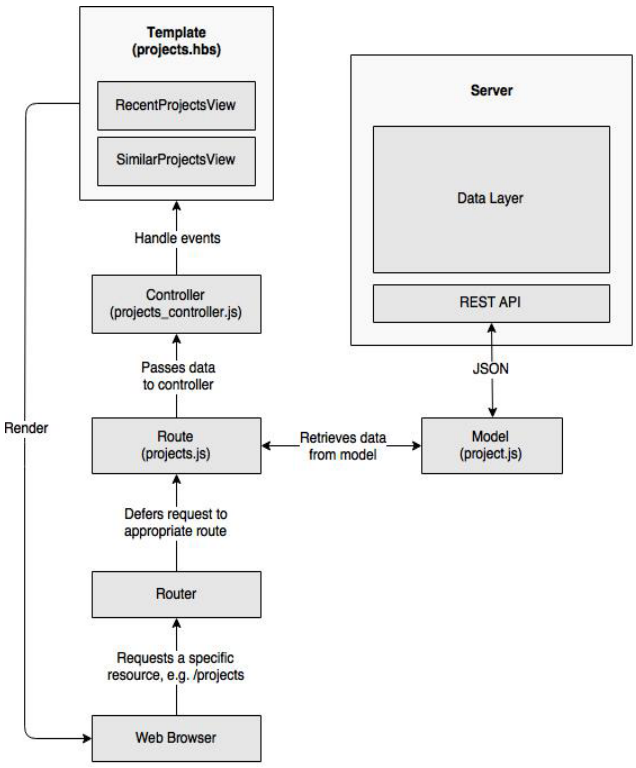

Figure 6. Request routing in Ember

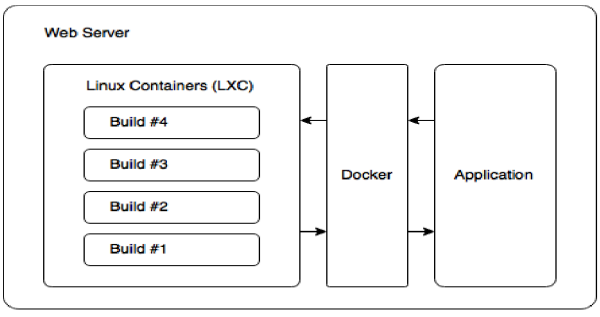

Figure 7. LXC containers (Application: Back end of Pons's web application).

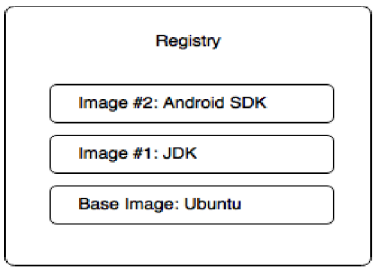

Figure 8. Docker Images.

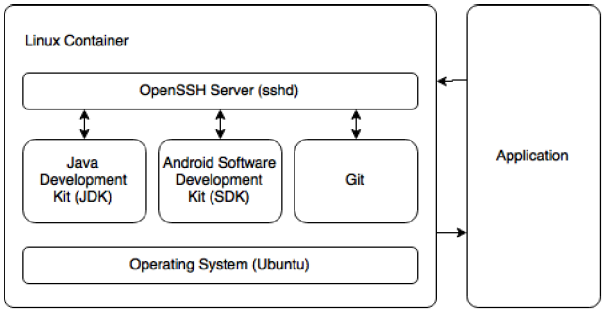

Figure 9. Communication between different components using OpenSSH.

3.5 Live Streaming and display

Delivering media as a stream is an alternative to downloading. Pons displays the Android emulator running inside a container by utilizing one of the live streaming tools called "X Persistent Remote Applications" (Xpra) [11]. Xpra is an Open Source Software tool that is used for directing a display from a remote host to a local machine. Xpra was installed on one of the Docker images, which the containers are built on. To display the streamed contents at the client side, Pons uses Xpra's HTML5 client This client establishes connection with the Xpra server installed in the container, receives buffer and images and decodes them as appropriate, before displaying them using HTML5 Canvas API.

3.6 Pre-release testing procedures

The basic procedures of building and streaming a mobile application to the browser of a client for the purpose of testing are as follows:

1. Client asks for a permission to get access to the repository containing the mobile application source code hosted on GitHub.

2. Pons grants the client an authentication token and saves it in its record, to keep track of this client.
3. Pons clone the repository that has the source code of the latest version of the Android application.
4. Upon obtaining the source code, Pons creates a Linux container (LXC) using Docker. The container will be based on a certain Docker image that can be accessed from a private Docker registry. The registery is hosted on the server that hosts the backend of Pons's web application.
5. Pons copies the repository into the container, creates an Android emulator to be used while building the application, starts an OpenSSH server and assigning a specific port number for it then run Ant [21] to build the mobile application.
6. Pons starts the Android emulator, and then install the Android application it using SSH commands.
7. Xpra live streams the emulator to the client. An HTML5 canvas element displays the video streaming. On top of that element, there is another canvas element used as an overlay for drawing annotations and leaving comments by the tester.

4 Conclusion and future work

Testing is critical to ensure the development of a high quality mobile application that meets the business requirements and achieve the customer satisfaction. Pre-release distribution for the purpose of manual testing is crucial. However, testing pre-release is tedious and time-consuming process, with many problems hindering application development. Along with that, communication remains a stumbling block between developers and testers. It is often broken, and both are usually isolated from the software under testing.
Pons was proposed to alleviate problems existing with pre-release mobile testing. Pons facilitates building mobile applications and running them directly in the browser. When an application runs and its environment is ready for manual testing, a peer-reviewing session starts, allowing developers and testers to engage in a meaningful feedback loop, in which they can submit feature requests, report bugs and software faults, and handle them as appropriate.
Currently, we are extending Pons to work for automated testing of mobile software as well.

References

1. Uber: https://play.google.com/store/apps/details?id=com.ubercab&hl=en.
2. AEMC: http://www.aemc.com/HTML-email/PR_Update_Clamp_On_Models_407_607/AEMC_PRU_407-607_0914.html
3. Empire Loan: https://play.google.com/store/apps/details?id=com.empireloan.cashcam&hl=en.
4. A. Sonderegger, S. Schmutz, J. Sauer, The influence of age in usability testing, applied ergonomic, vol 52, PP. 291-300, (2016).
5. M. Masood, M. Thigambaram, The Usability of Mobile Applications for Preschoolers, 7th World Conference on Educational Sciences, (WCES-2015), NovotelAthens Convention Center, Athens, Greece, (2015).
6. D. Zhang and B. Adipat. Challenges, methodologies, and issues in the usability testing of mobile applications. International Journal of Human-Computer Interaction, PP. 293-308, (2005).
7. D. Amaltano, A. R. Fasolino, P. Tramontana, S. Decarmine, and A. Memon, Using GUI ripping for automated testing of android applications. In Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, pages 258-26, (2012).
8. C. Hu and I. Neamtiu. Automating GUI testing for android applications. In Proceedings of the 6th International Workshop on Automation of Software Test, PP. 77-83, (2011).
9. J. Turnbull. The Docker Book: Containerization is the new virtualization. James Turnbull, (2014).
10. J. Loeliger and M. McCullough. Version Control with Git: Powerful tools and techniques for collaborative software development. "O'Reilly Media, Inc.", (2012).
11. Xpra live streaming: https://www.xpra.org/, (2015).
12. D. Flanagan. JavaScript: the genitive guide. O'Reilly Media, Inc., (2006).
13. Ruby on Rails Guides: http://guides.rubyonrails.org/, (2015).
14. N. Nurseitov, M. Paulson, R. Reynolds, and C. Izurieta. Comparison of Json and Xml data interchange formats: A case study. Caine, PP. 157-162, (2009).
15. Docker: Https://www.docker.com/.
16. D. Merkel. Docker: lightweight linux containers for consistent development and deployment. Linux Journal, (2014).
17. Virtual machine KVM: www.linux-kvm.org.
18. Linux Containers LXC: https://linuxcontainers.org/, (2015).
19. Openssh: www.openssh.com
20. D. Berube. Practical Ruby Gems. Apress, (2007).
21. Ant: ant.apache.org/
22. J. Humble and D. Farley. Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation. Addison-Wesley Professional, 1st edition, (2010).