

Visualizing Branches and Metrics of Version Control Systems on Mobile Devices

Will Boyd*, Nicholas Diliberti, and Haeyong Chung

Department of Computer Science, University of Alabama in Huntsville

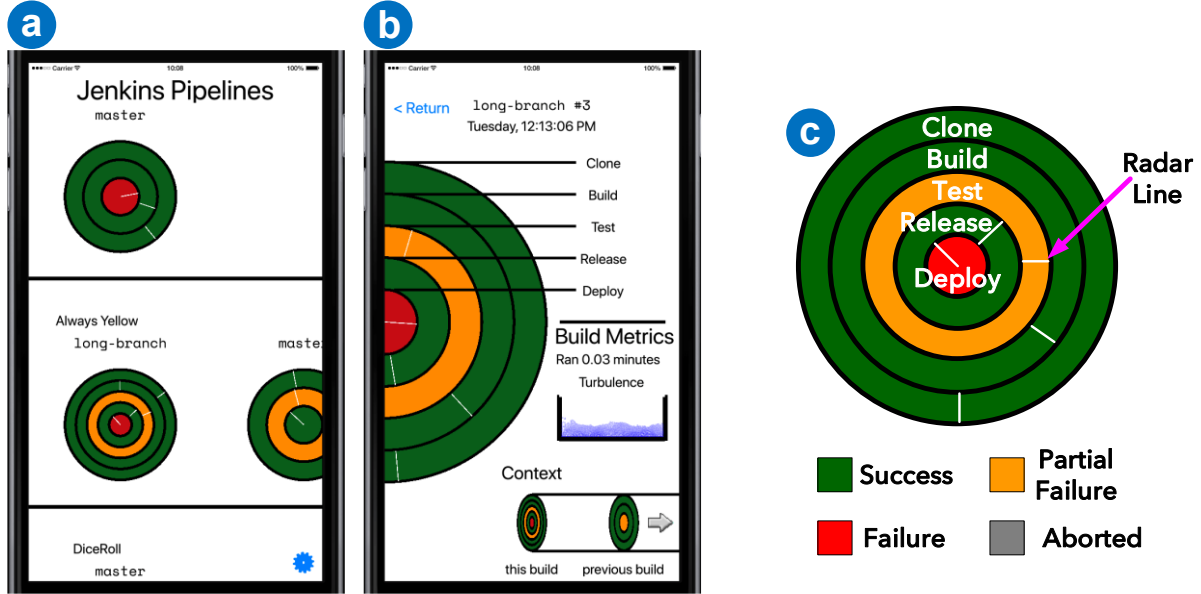


Fig. 1. Jenkins pipeline visualization for mobile devices. (a) The Landing view. (b) The Branch view. (c) The branch circle representation.

ABSTRACT

In this work, we present the Jenkins Pipeline Visualization. This mobile data visualization is designed to help developers explore a set of version control repositories tested by a Continuous Integration/Continuous Deployment (CI/CD) server and analyze their build metrics on mobile phones.

Keywords: Mobile visualization, agile development, automated testing, continuous integration / continuous deployment.

1 INTRODUCTION

Increasingly, agile software projects use Continuous Integration/Continuous Deployment (CI/CD) practices. CI/CD platforms such as Jenkins, TeamCity, and Travis monitor version control platforms, often based on Git. When a developer commits code to the repository, the CI/CD platform pulls the code from version control and runs a *pipeline*—a series of predefined stages with pass, fail, or partial success states. These pipelines are configured and by the product’s developers and managers. Pipeline stages include building binaries, performing unit tests, uploading build artifacts to a binary repository, and deploying

built products to a staging ground or test system.

After a pipeline is done, developers review results for debugging purposes and for metrics analysis, employing CI/CD user interfaces. There are different ‘depths’ or factors for a user interface to support such tasks. One factor is the technical level of a user. *Developers* that write code are very well acquainted with the low-level details of the code. On the other hand, *business owners* may have never written code professionally, or may be unfamiliar with the product’s technical details. For this reason, varying levels of information should be presented by a user interface. In addition, developers can also benefit from high-level views, for example when in a meeting, or acting as a *triage team*, whose job is to decide where to funnel issues, but not actually solve the issues themselves. Visualizing failure-log information in multiple views is thus beneficial to every class of user.

The developers of Jenkins, which is one of the most commonly used CI/CD systems, have invested significant work into their desktop web interface. However, the interface focuses on showing textual logs and requires on-hand, technical information to parse.

The Jenkins Pipeline Visualization (JPV) is designed to help users explore a set of version control repositories tested by a Jenkins CI/CD server on mobile phones (Figure 1). As mobile devices become more ubiquitous, developers would be well-served by a mobile platform that visualizes the status of their pipelines, especially when monitoring tests remotely. To display to various mobile screens more efficiently, JPV employs novel visualization approaches based on animations.

Related work with JPV includes research on visualizations for code repositories [1, 2]. However, we think that using mobile

* wkb0004@uah.edu

devices and visualizations will open up new opportunities in ubiquitous analysis of version control systems [3, 4, 5]. In contrast to the prior research that is focused on general visualization platforms, our work specifically investigates supporting CI/CD visualization on mobile devices anywhere and anytime.

2 VISUAL REPRESENTATION AND VIEWS

JPV focuses on visualizing multiple branches as a diverse collection of stages where each branch runs in a uniform and understandable manner. The system provides two primary visualizations: a high-level view (the Landing view) of all pertinent pipelines, and a deep-level view (the Branch view) of a selected pipeline to give a technical user the information needed to continue debugging on a desktop system.

Visual Representation for the Branches: The atomic unit of the JPV application is the branch. In a Jenkins pipeline, each branch correlates to a branch in the version control system (e.g. Git). Each branch has a collection of runs, and each run visualizes the results for a set of user-designed tests.

As shown in Figure 1c, JPV visualizes sets of branches as sets of concentric rings, each ring representing a test stage in the pipeline. Rings are colored with the same colors used by Jenkins: green for success, red for failure, yellow for partial failure, and grey for an aborted stage. Each stage has a radar line that rotates around the circle, its speed proportional to the percentage of the build time it consumed.

Landing View: When users open the app, they are taken to the Landing view (Figure 1a). Each repository has its branches visualized in carousel view. Each branch shows its most recent build, complete with radar line. The user can scroll through the repositories and branches, then tap on a desired pipeline.

Branch View: By selecting a branch representation in the Landing view, the user is taken to the Branch view (Figure 1b). The branch view shows detailed visualization of the selected branch as a larger half-circle on the side of the screen. Each stage of the pipeline has a callout with the name, and a link to the branch results is shown.

To represent branch information, JPV employs animation features, which have been proven effective in mobile visualizations. [6]. In addition to radar lines visualizing build time, the turbulence visualization represents the ratio of failed builds to total builds. Turbulence is visualized as a simulated body of water. As a build has more consecutive failures, the water becomes more turbulent, and as a build has more success, the water becomes calmer. We use the last five builds to measure the ratio of failed recent builds to successful recent builds. Thus, the developers can understand their build status quickly. This view is shown in Figure 2.

At the bottom of the screen, a historical context for the build is shown, giving the user a view of the current build, the next build (forward in time), and the last build (backwards in time).

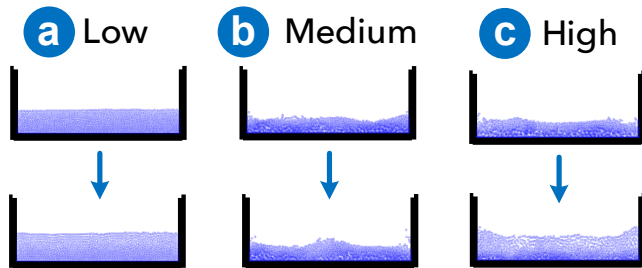


Fig. 2. Examples of the build turbulence animation: (a) 1/5 of the last builds failing, (b) 3/5 of the last builds failing, and (c) 5/5 of the last builds failing. As a build has more consecutive failures, the water becomes more turbulent.

3 USAGE SCENARIO

To illustrate the potential of JPV in CI/CD practices, we apply it to hypothetical mobile visualization use case for a software company. Alice is a developer with a company, while Bob is the release manager in Alice's department. Alice is well acquainted with the inner workings of the product, while Bob is only generally knowledgeable about the product's components and how they connect. The company maximizes its CI/CD use in both build and releases.

Bob is concerned with the final state of the product. As release manager, he is responsible for making sure the product is in a state usable by the customer. When a build fails, turnaround time is important, especially as they approach release dates. While he is away from his desk, he needs to monitor the status of the repositories and immediately assign developers to any emergent errors. The Landing view in the app provides this high-level data, so that when a build fails, he can quickly triage the situation.

Alice has meetings and personal errands that call her away from her desk. However, Alice also commits code and watches the build system pull her contributions through increasingly larger builds. While she is away from her desk, she can use the landing page to watch a change propagate through various pipelines. If the build fails, she can use the Branch view to investigate it. Here, she can see both the stage in error and its recent history. This is useful in meetings with other developers and management where the readiness or general status of the product is being discussed. The high-level overview provided by the Landing view, along with the general overview provided by the Branch view, allows her and other developers to have informed conversations while phoning in to meetings or away from their desks in conference rooms.

4 CONCLUSION AND FUTURE WORK

We believe that JPV can empower managers to monitor the state of their product and developers to chart the course of their builds. The CI/CD platforms have an unfulfilled need for an application to aid users in their everyday work activities. JPV can also be easily generalized to other code repository systems. Future work should include investigating the visualizations shown here on different sizes of mobile devices. A future version might also consider turning the pipeline historical context view into a scrollable view—by “scrubbing” the pipeline left and right, the user would move the visible pipeline rendering forwards and backwards through time.

REFERENCES

- [1] J. L. C. Izquierdo, V. Cosentino, B. Rolandi, A. Bergel, and J. Cabot. GiLA: GitHub label analyzer. In *Proc. of IEEE SANER 2015*, pp. 479-483, 2015.
- [2] Z. Liao, D. He, Z. Chen, X. Fan, Y. Zhang and S. Liu. Exploring the Characteristics of Issue-Related Behaviors in GitHub Using Visualization Techniques. *IEEE Access*, vol. 6, pp. 24003-24015, 2018.
- [3] N. Elmqvist and P. Irani. Ubiquitous analytics: Interacting with big data anywhere, anytime. *Computer*, 46(4), pp. 86-89, 2013.
- [4] B. Lee, M. Brehmer, P. Isenberg, E. K. Choe, R. Langer, and R. Dachsel. Data visualization on mobile devices. In *Extended Abstract Proc. of ACM CHI 2018*, pp. 1-8, 2018.
- [5] B. Watson and V. Setlur. Emerging research in mobile visualization. In *Tutorial Proc. of ACM MobileHCI 2015*, pp. 883-887, 2015.
- [6] M. Brehmer, B. Lee, P. Isenberg, and E. K. Choe. A comparative evaluation of animation and small multiples for trend visualization on mobile phones. *IEEE Transactions on Visualization and Computer Graphics*, 26(1), pp. 364-374, 2019.