International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395 -0056
Volume: 03 Issue: 11 | Nov -2016 www.irjet.net p-ISSN: 2395-0072

Perceptual Difference for Safer Continuous Delivery

Ankit Ramakrishnan, Dr. Manjula R

Undergraduate, School of Computer Science and Engineering, VIT University, Vellore, Tamil Nadu, India
Professor, School of Computer Science and Engineering, VIT University, Vellore, Tamil Nadu, India

---

Abstract - *Perceptual Difference is a proposed continuous integration tool to enable safer Continuous Delivery. Existing Continuous Delivery tools used for automated testing of Web Applications are inherently dependent on code based testing that uses algorithmic testing and is devoid of human perception. Perceptual Difference combines concepts of Computer Vision and CI to enable recognition of UI/View based changes, assisting human testers to check development branches with added scrutiny before deploying the application.*

Key Words: Continuous Delivery; Perceptual Difference; Automated Testing

1. INTRODUCTION

Continuous Delivery and Continuous Integration are common Software Engineering Paradigms used in the Software Industry. They rely on a set of processes and workflows that enable multiple programmers or developers to access the Codebase and submit changes and revisions. Continuous Delivery (CD) is a superset of Continuous Integration (CI). Every change on the Codebase is immediately deployed to production in CD after terse and often very stringent testing and approval mechanism. Most Penultimate processes in CD Involve testing and approval. As a result of CI practices, Testing is majorly automated and requires little to no human intervention. Testing is usually followed by Quality Assurance (QA) which entails approval of the currently staged branch of the Codebase. Perceptual Difference (Perceptual Diff) is intended to reduce this QA time by assisting in identifying changes in the UI. This enables faster and safer CD.

The main aim of this methodology is to increase reliability of automated tests. This will greatly increase speed and accuracy of user acceptance tests, thus, resulting in faster deployment rates. Faster release cycles are essential in a market with increasing number of competitive agents that are forever on the lookout for stumbles that a vendor makes in an attempt to gain market share from their competitor's loss.

We shall focus on key aspects of perceptual image difference, CD integrations and other metrics to determine the efficiency of the proposed tool.

2. CONTINUOUS DELIVERY

Continuous Delivery as defined by the Agile Alliance is an extension of the Continuous Integration methodology (CI) that tries to reduce the cycle-time (lead time), which is the time taken for a line of code written in development to be used in the live version of a product which is targeted to users. CD methods often involve the creation of Delivery and Deployment Pipelines which are a series of integrations from the point of authoring code to the final deployment to a production environment.
These Integrations form the basis for Continuous Integration (CI). In multi author projects that have a large number of developers that have access to the development branch of the project, it is necessary to have integration mechanisms in place such that no single merge can compromise the state of the production branch.

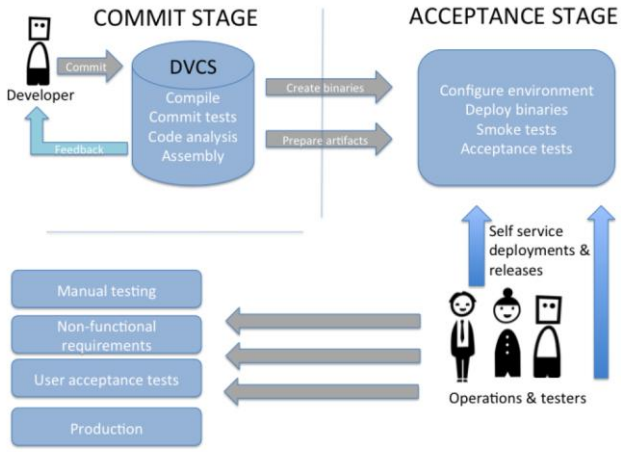


Fig-1: Anatomy of the Deployment Pipeline [1][2]

2.1 Continuous Integration

CI in the above scenario become absolutely necessary. CI entails that the developers on the developer branch merge their code to the main repository at regular intervals. After merging this code a variety of build and test automation tasks are performed to ensure build integrity. A CI pipeline is usually centrally coordinated with a CI Server. Every revision of the codebase triggers a central build test for that revision. If approved the revision is merged with the staging/production branch. If the tests or the build process

fail the revision is marked as a "fail" and the developer is notified.

2.2 Build Automation

Build Automations is an integral aspect of CD methodology. It refers to self-contained environments that perform tasks of building from source in a deterministic and consistent manner.

Developer Local Environments do provide a simple benchmark for building projects but are woefully inconsistent across platforms. They thus don't give us a global picture of build failure with respect to the live deployment. Users may have different systems and configurations that might react differently to the product.

Automation ensures a build consistency that is not observed on Developer Machines. A build automation agent or server is a central server that can be –

2.2.1. On Demand – Build is performed when required by the developer and has to be manually run.

2.2.2. Scheduled – The build process is scheduled ahead of time, similar to regular merges in the Source Code Repository.

2.2.3. Triggered – Triggered automation is the most powerful form of build automation as it is triggered whenever there is a change on the main SCM.

Build automation tools are necessary for CD to produce reliable binaries ready for production.

2.3 Test Automation

Testing a build is essential before deployment. However, a lot of regressing and unit tests are highly time consuming and can cause delays in the lead time estimates of production. This can however be avoided by automating testing by utilizing the repetitive nature of testing. Unit tests can be written prior to testing and run in bulk by a Test Automation Server. The server is responsible for providing stakeholders with data associated with test completion or failure.

Most popular TA frameworks provide a simple scripting language to control the tests themselves and architecting them. Continuous Testing is a subset of TA that is similar to CI and CD and involves running testing tasks as part of the CD pipeline and thus getting immediate feedback on the risks associated with the current build.

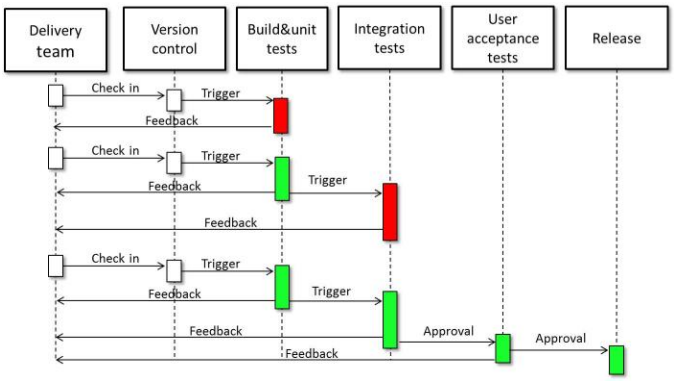


Fig- 2: Flow diagram for the CD Pipeline [3]

3. Current Tools

3.1 Jenkins

Jenkins is a very popular and powerful CI Server that is used by almost all industry giants to orchestrate their software delivery pipeline.

Jenkins was originally part of the Hudson Project at Sun Microsystems but later due to issues with the company Kohsuke Kawaguchi released the product separately as the Jenkins CI Server. It is written entirely in Java and is completely open source. The Jenkins Automation Server Boasts of powerful integrations with most databases and SCMs such as Git or SVN. It can be integrated with build tools like Maven or Gradle. As a result of the open source nature of the project Jenkins has a rick plugin ecosystem that is continuously updated and maintained by its contributors. It can be extended and modified as and when needed by the organization. It has a friendly web-based GUI interface to enable on the fly configurations with little to no code experience.

Error reporting in Jenkins is very user friendly and provides users with inline help.

3.2 Gradle and Maven

Gradle is an open source build automation tool written in Java and Groovy. It was designed for multi project builds that have large dependency trees that can be hard to keep track if build is performed manually across platforms and machines. Gradle uses acyclic graphs to emulate build tasks and each subsequent task depends on the output of the last. It was originally a build tool for Java, Groovy and Scala. Currently it supports 60 different programming languages and environments including but not limited to C/C++, Python, Android, and iOS. Gradle is proficient in handling multiple dependencies that are described in a variety of formats such as Ant, Maven or Ivy.

The Acyclic nature of the tool helps incremental builds that can result in sequential build artefacts that can later be sent to distributed test automation servers. Build Caching is

another powerful feature of Gradle that entails building only parts of the project that have cone revisions or change.

Maven or Apache Maven is another build automation tool used for projects. Similar to Gradle it supports integrations with local or online artefact repositories. Maven describes the configuration of the project in XML format called the POM or the Project Object Model. The POM keeps track of the plugins and libraries used by the project and necessary for building.

3.3 Selenium

Selenium is a web browser automation tool that is used for testing Web Applications by emulating user interactions like touches and input. Selenium is written in Java and utilizes a WebDriver. The WebDriver starts browser instances with the desired application running on it and sends commands to it depending on the test specifications that can be provided in a variety of languages. Selenium has bindings in almost every major language like C/C++/Python.

Table -1: Sample Table format

4. WEB APPLICATIONS

Web applications are currently on the rise and undoubtedly the most popular form of applications. They utilize the browser as a deployment vector and users view and interact with these applications via the browser itself without the need of downloading an external desktop application which is usually bulkier. Web apps are known for their increased accessibility and cross-platform nature. Different applications are not needed to be made for different platforms as in the case of Desktop Applications where different OSes require different binary formats, libraries and languages. Web applications cut through this clutter by streamlining the deployment strictly to web browsers.

Their user acceptance is usually guided by interfaces and front end interactions. Most popular websites invest a lot of time and capital on designing and modifying their interfaces so as to attract more users and/or customers. The User Interface Design and Development is a key component in the software development stage. It involves specifying the layout, the design and the interactivity of the web application.

Common tools used to accomplish this are –

4.0.1. HTML for layout specification

4.0.2. CSS for designing the appearance

4.0.3. JavaScript for enabling interactions

4.1 Development

Modern web applications use a variety of architectures of which the MVC architecture is the most common. Front end frameworks such as Angular, React, EmberJS and JQuery are used to make rich Single Page Applications.

4.2 Testing

With the introduction of Selenium WebDriver and similar technology writing tests for web applications has been immensely simplified. The developer specifies unit tests that need to be performed on the application and the Selenium WebDriver automates these Use Cases and provides the output to the tester.

While Selenium only caters to Functional and End to End testing. Web apps need to undergo the following testing methodologies as well –

4.2.1. Functional Testing – This entails that all the hyperlinks, forms, buttons, database connections are working properly in the web application. Testing links would include testing all the internal link jumps, external link jumps, check email links and check for orphan pages.

4.2.3. Usability Testing – This is the test which ensure that navigation in the web application is according to the set of rules provided by the developer and works accordingly. Usability includes navigation controls and other application components that perform view changes.

4.2.4. Interface Testing – This test is to ensure the interface between the web server or the backend is working in tandem with the front end or the view. It makes sure that error messages are correctly reported at the backend in case anything goes wrong.

4.2.5. Compatibility Testing – This test makes sure the app performs consistently across browsers ad platforms. It keeps note of JavaScript incompatibilities of certain browsers and tests these specific functions of the web app on the specific browser versions.

4.2.6. Performance Testing – This test is divided into load ad stress testing. Load testing involves assessing if the application can handle large quantities of user traffic and analyzing its limits. Stress testing takes the application to its specified limits and observes how the application recovers from a stress case scenario like a crash for example.

5. DRAWBACKS OF CURRENT FRAMEWORKS

Current web frameworks like Selenium do an excellent job of automating most repetitive tasks performed while testing a web application like checking link status and navigation but they are essentially blind to the interface.

Selenium does have support for CSS Selectors but this does not enable it to directly observe or understand the changes that occur in a page as a result of CSS code change. It also is content agnostic. This means typos and omissions in the content are not given priority or are not reported entirely.

For example, in the landing page of a major company if the company copywriter decides to make some changes to the "about" page then the content change of this revision in code will no trigger any fails in the test automation stage. The tester can manually write content tests for each piece of content but this would be too inefficient and time consuming.

Alternatively, if the designers of the web page decide to change the orientation or alignment of components then they would effectively have to change the CSS code. These minor changes would define pixel positions and sizes. Again, test frameworks are view agnostic and only test rendered DOM code. This means any view specifications would ha ve to be hard coded in the test suite and as mentioned above would be woefully time consuming. The goal of automation is to reduce time needed for testing applications. The above scenarios have made certain that current automation technologies cannot identify or differentiate minor interface changes. That is, they are view agnostic.

Perceptual Difference is the proposed solution which enables the tool to see the changes that occur on a page visually and flags the page as high priority for manual QA and User Acceptance Testing.

6. IMAGE DIFFERENCE

To analyze changes in a given source image it is sufficient to calculate the image difference of this image with the target image given the compression algorithm is lossless.

Image Difference is a simple Image Processing technique that involves subtracting one image from the other. This process is very useful in identifying changes in an image.

For example,

Fig- 3: Original Wikipeida Homepage

Fig- 4 : Changed Wikipedia Homepage

Fig- 1: Image Difference

The above example shows how minor changes in the homepage are not easily registered visually when viewed by the manual tester but can be clearly identified once image difference is applied.

The first image has a glow around the text field and also has Deutsch as the second highest number of articles. Whereas the modified homepage has Hindi listed as the same. This change is not very obvious to a manual tester. However to a computer after applying the Image Difference algorithm the change is very apparent and it can be at once said that there has been a change and the changes can be tagged allowing the manual tester to test the page with more scrutiny.

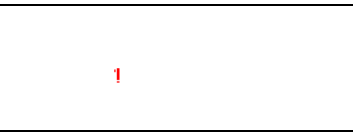
7. P-DIFF

pDiff is the proposed tool to integrate image differences into the CD pipeline. As interface changes are hard to manually keep track of via code, pDiff adopts the more intuitive solution of storing screenshots of the live and staged versions. The screenshots vary depending on the URL, build number, and JavaScript interaction.

After the CI Server has green flagged the current revisions and a new build is generated, the new build is immediately captured by the pDiff tool and individual screenshots of all the screens are taken. All the URLs in the config file are traversed and the images are stored with the release number. These image differences are marked with a percentage of difference which is a reference to the amount of apparent change in the product. The complete pDiff review of the current release is then available for the Manual Tester, QA Personnel to check. With visual cues provided to help the tester, it is guaranteed that even minor mistakes or changes in these applications will stand out and garner more attention of the tester.

The revisions are stored in the given format –

Table- 1: pDiff Representation Format

Id	Image
Reference	
Run	
Diff	

7.1 Establishing a Baseline

To use pDiff effectively or to integrate it into the current development methodology it is essential to add a baseline to the currently built application. This entails pDiff crawling through the live version and storing the screenshots with the date and time recorded for easy indexing.

7.2 Creating a New Release

Each time a new release is created post the CI Build Automation Task, the CI can trigger the pDiff server to start crawling through the new release and storing the screenshots along with the difference screenshots.

7.3 Manual Approval

The Manual Tester can now access the pDiff web application to check the status and manually approve or reject each difference marked by the tool.

7.4 Marking the Release

The tester can now mark the entire release as good or bad depending on the types of differences. Once the release is marked as good the release can be automatically deployed as it has been approved and meets the standards of the application.

The pDiff application maintains a central server that uses MySQL for database storage but can alternatively also use SQLite. pDiff organizes the releases based on the build ID that is released by the CI Server. A new build can also be created using the API server's UI. A build refers to a binary that is ready to be deployed on a production server. As each release can be repeated multiple times, a history of each released is also maintained by the server and changes can be rolled back if not required.

The visual difference of multiple page is termed as a test run and there can be different test runs depending on the nature of the application. The life-cycle of the release is –

- Create – A new release is created
- Receive – The release is waiting for tests to run
- Process – Tests have been run but some other tasks are yet to finish.
- Review – The Manual Tester is currently reviewing the changes and differences.

7. CONCLUSIONS

In conclusion it is safe to say that the pDiff tool can be a valuable addition to the CD pipeline as it adds increased reliability to manual testing. Perceptual Difference applications can also be readily integrated with current CI tools like Jenkins. pDiff server and API can be triggered directly with Jenkins plugins.

As companies increase in size and in terms of the number of developers assigned to the project it becomes easier to let small errors slide that are not reported even by the automated error reporting mechanisms. These errors reduce reliability and trust of the product. This in turn increases the lead time associated with a project. Overhead time associated with testing can be significantly reduced by completely automating even the smallest of details that can be done with the help of a program. As lead time reduces, programmer efficiency and productivity also increases as a result of increased happiness on having code going live in relatively frequent intervals. At the end of the day, a happy programmer is a productive programmer.

ACKNOWLEDGEMENT

The Authors thank the open source community for supporting and helping developers in all fields of the industry. We would also like to thank Bret Slatkin for his talk at the Velocity Software Development conference without whom this paper would not have been possible.

REFERENCES

[1] S. Neely and S. Stolt, "Continuous delivery? easy! just change everything (well, maybe it is not that easy)," in Agile Conference (AGILE), Aug 2013, pp. 121–128

[2] Humble and D. Farley, Continuous delivery: reliable software releases through build, test, and deployment automation. Pearson Education, 2010.

[3] Timo Lehtonen, Sampo Suonsyrjä, Terhi Kilamo, and Tommi Mikkonen in Defining Metrics for Continuous Delivery and Deployment Pipeline SPLST 2015

[4] A. Maruf Aytekin, Release Management with Continuous Delivery:A Case Study in International Journal of Social, Behavioral, Educational, Economic, Business and Industrial Engineering Vol:8, No:9, 2014

[5] Leigh Garrett Amy Robinson. Spot the Difference! Plagiarism identification in the visual arts, unpublished.

[6] Amruta Kumbhar, Madhavi Shailaja & Ravi Shankar Anupindi, Getting Started with CI in software development, White Paper, InfoSys 2015

[7] J. Humble, C. Read, and D. North, "The deployment production line," in Agile Conference. IEEE, 2006, pp. 6–pp

[8] P. Debois, "Devops: A software revolution in the making," Cutter IT Journal, vol. 24, no. 8, 2011.