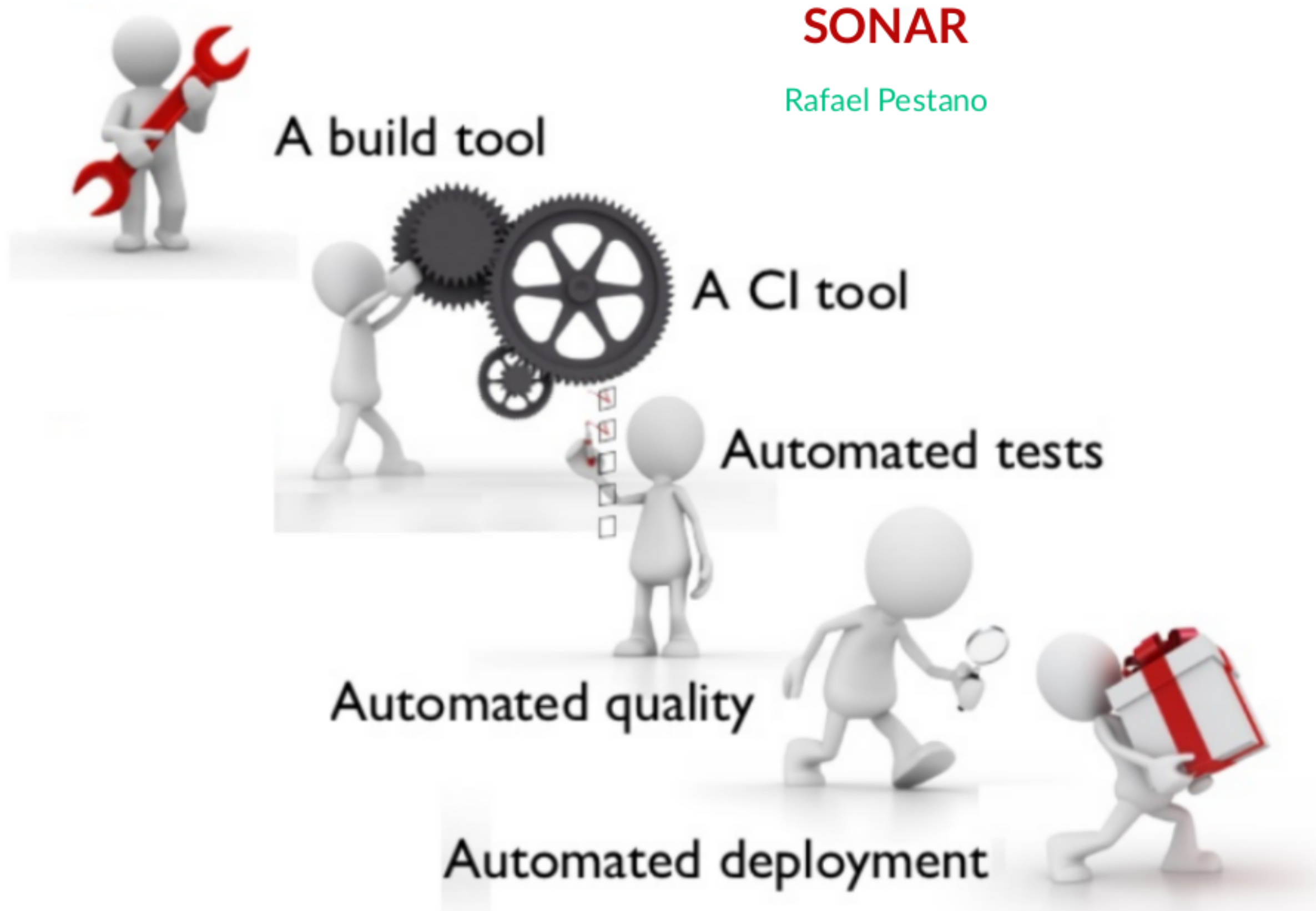


JAVAEE PIPELINE AS CODE USING JENKINS, DOCKER AND SONAR

Rafael Pestano



RAFAEL PESTANO

Software Engineer at PROCERGS

- @realpestano 
- @rmpestano 
- Blog 



See this presentation in [HTML](#) here.

CONTINUOUS DELIVERY

"A software strategy that enables organizations to deliver new features to users as fast and efficiently as possible"

CONTINUOUS DELIVERY

"A software strategy that enables organizations to deliver new features to users as fast and efficiently as possible"

- **Be ready** to delivery in production at any moment!

WHY?

"... If it hurts, do it more frequently, and bring the pain forward."

— Jez Humble

GOALS

- Reduce the risk of delivering
- Create a well known delivery process/cycle
- Make release process painless and without surprises
- Be ready to go to production at anytime

Change

Required.
Often feared.
Why?



<http://www.flickr.com/photos/20408885@N03/3570184759/>

DEPLOY TO PRODUCTION



PRINCIPLES

PRINCIPLES

- Each commit/push creates a release candidate

“The longer you delay, the worse(exponentially) the problem becomes” [Neal Ford - Director at ThoughtWorks]

PRINCIPLES

- Each commit/push creates a release candidate

“The longer you delay, the worse(exponentially) the problem becomes” [Neal Ford - Director at ThoughtWorks]

- Heavily based on automation

PRINCIPLES

- Each commit/push creates a release candidate

“The longer you delay, the worse(exponentially) the problem becomes” [Neal Ford - Director at ThoughtWorks]

- Heavily based on automation
- Automated tests are primordial

PRINCIPLES

- Each commit/push creates a release candidate

“The longer you delay, the worse(exponentially) the problem becomes” [Neal Ford - Director at ThoughtWorks]

- Heavily based on automation
- Automated tests are primordial
- Continuous and fast feedback (from end user as well from your release process)

PRINCIPLES

PRINCIPLES

- Continuous improvement

PRINCIPLES

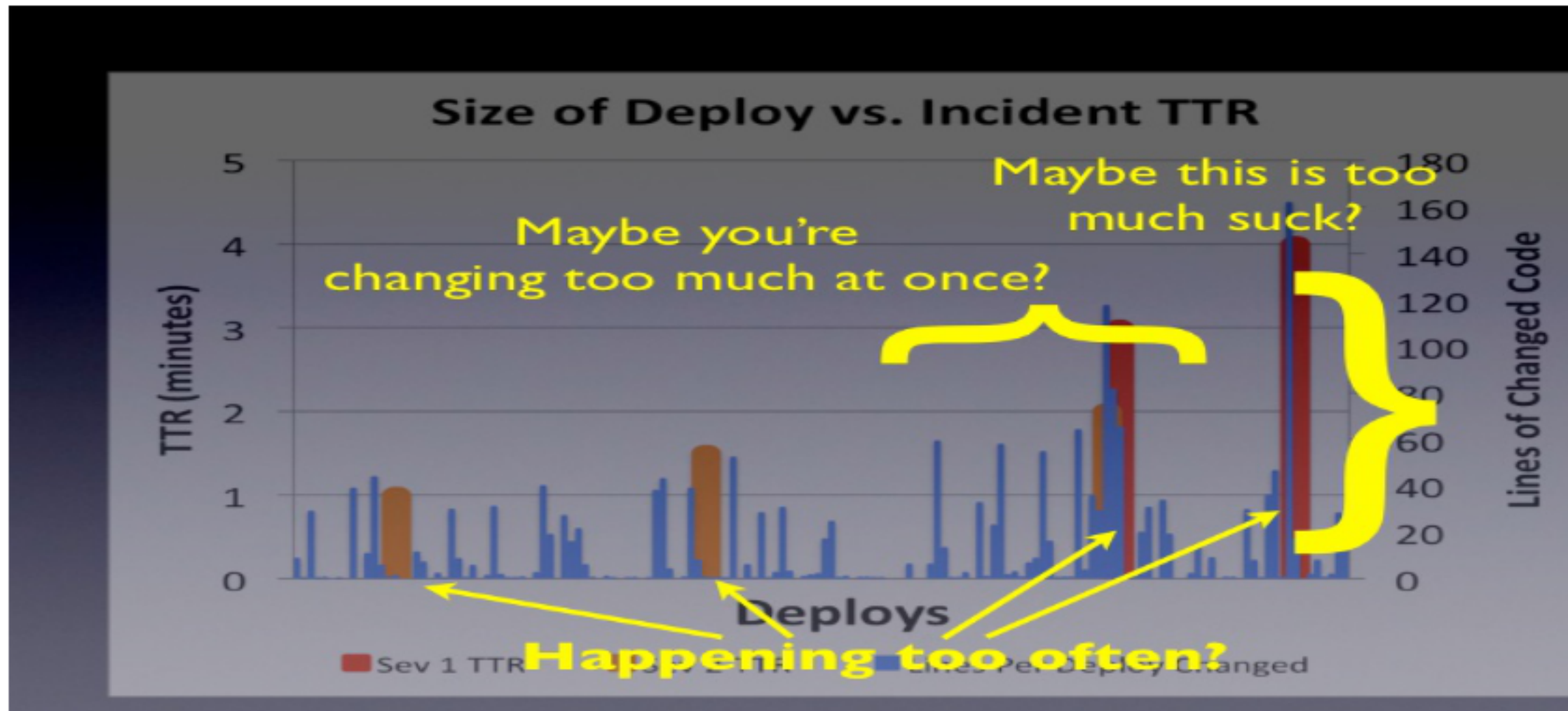
- Continuous improvement
- Colaboration, everyone is responsible for the release process (DEV, QA, OPs...)

PRINCIPLES

- Continuous improvement
- Colaboration, everyone is responsible for the release process (DEV, QA, OPs...)
- Measurable progress
 1. How many builds have failed?
 2. In which stage it failed?
 3. How long it takes to go to production?

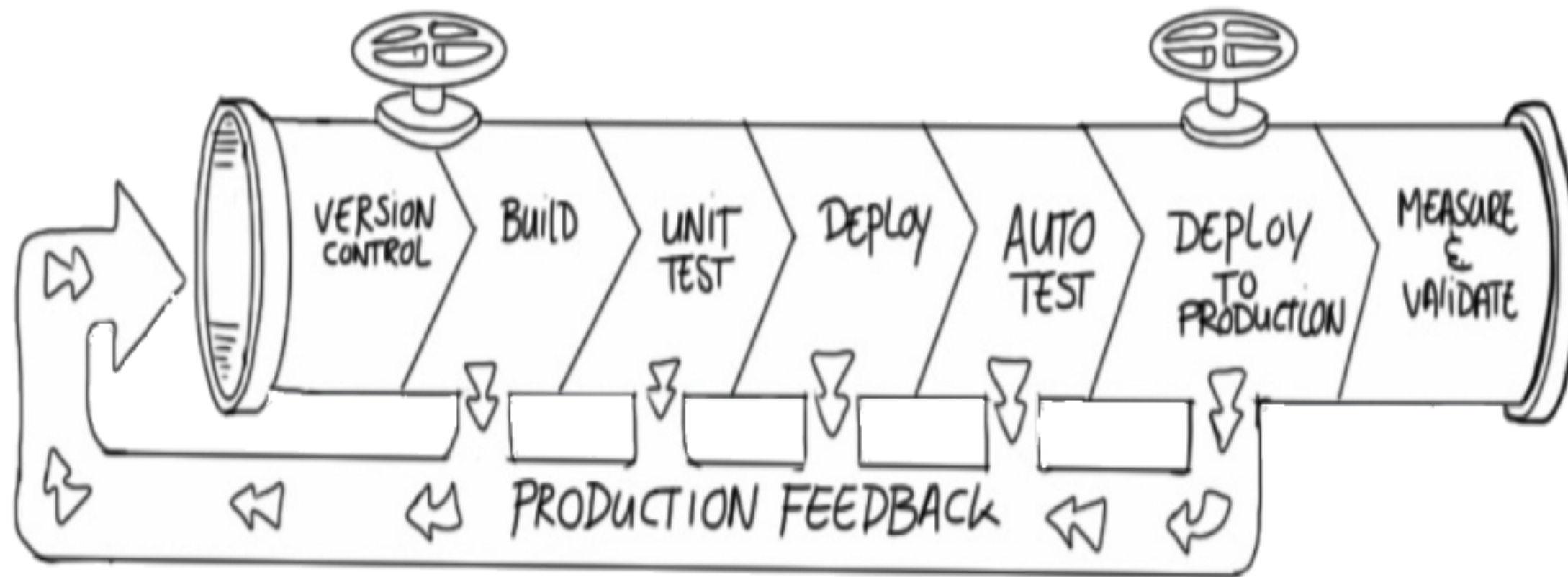
CONSEQUENCES

- Less severity and frequency of failures related to a release
- Reduced time to recovery from failures (MTTR)



DEPLOYMENT PIPELINE

"... A pipeline is a set of stages to bring functionality from developers to end users"



JENKINS 1.X PIPELINE

The screenshot displays the Jenkins 1.X Pipeline interface. At the top, the Jenkins logo is on the left, and a search bar, user name (Trevor Quinn), and log out link are on the right. Below the header, the breadcrumb "Jenkins > Build Pipeline" is shown, along with a "DISABLE AUTO REFRESH" link. The main section is titled "Build Pipeline" and contains a toolbar with icons for Run, History, Configure, Add Step, Delete, and Manage. Below the toolbar, four pipeline runs are listed, each with a "Pipeline" button and a sequence of stage buttons connected by arrows. The stages are color-coded: green for successful completion, red for failure, and blue for in-progress or pending.

Pipeline	Stage	Status	Time	User
#87	#87 Commit	Success	Dec 18, 2014 2:07:08 PM	developer
	#102 Acceptance	Success	Dec 18, 2014 2:12:43 PM	
	#48 UAT	Failure	Dec 18, 2014 2:13:23 PM	tester
	Production	Pending		
#86	#86 Commit	Success	Dec 3, 2014 9:45:11 AM	developer
	#99 Acceptance	Success	Dec 3, 2014 10:13:16 AM	
	#45 UAT	Success	Dec 3, 2014 10:15:01 AM	
	#28 Production	Success	Dec 3, 2014 10:16:41 AM	
#85	#85 Commit	Success	Dec 3, 2014 9:42:11 AM	developer
	#97 Acceptance	Failure	Dec 3, 2014 9:42:31 AM	
	UAT	Pending		
	Production	Pending		
#84	#84 Commit	Failure	Dec 3, 2014 9:39:31 AM	developer
	Acceptance	Pending		
	UAT	Pending		
	Production	Pending		

JENKINS 2.X PIPELINE

Declarative Pipeline Syntax 1.0 is now available

Published on 2017-02-03 by Patrick Wolf

 Tweet

pipeline blueocean

I am very excited to announce the addition of [Declarative Pipeline syntax 1.0](#) to [Jenkins Pipeline](#). **We think this new syntax will enable everyone involved in DevOps, regardless of expertise, to participate in the continuous delivery process.** Whether creating, editing or reviewing a pipeline, having a straightforward structure helps to understand and predict the flow of the pipeline and provides a common foundation across all pipelines.

Pipeline as Code

Pipeline as Code was one of the pillars of the Jenkins 2.0 release and an essential part of implementing continuous delivery (CD). Defining all of the stages of an application's CD pipeline within a `Jenkinsfile` and checking it into the repository with the application code provides all of the benefits inherent in source control management (SCM):

- Retain history of all changes to Pipeline
- Rollback to a previous Pipeline version
- View diffs and merge changes to the Pipeline
- Test new Pipeline steps in branches
- Run the same Pipeline on a different Jenkins server

JENKINS 2.X PIPELINE

JENKINS 2.X PIPELINE

- Described in a very easy and powerful DSL

JENKINS 2.X PIPELINE

- Described in a very easy and powerful DSL
- Lives on source code (versioning)

JENKINS 2.X PIPELINE

- Described in a very easy and powerful DSL
- Lives on source code (versioning)
- Reuse with **shared libraries**

JENKINS 2.X PIPELINE

- Described in a very easy and powerful DSL
- Lives on source code (versioning)
- Reuse with **shared libraries**
- Everything in one place (Jenkinsfile)

JENKINS 2.X PIPELINE

- Described in a very easy and powerful DSL
- Lives on source code (versioning)
- Reuse with **shared libraries**
- Everything in one place (Jenkinsfile)
- Recovery from restarts

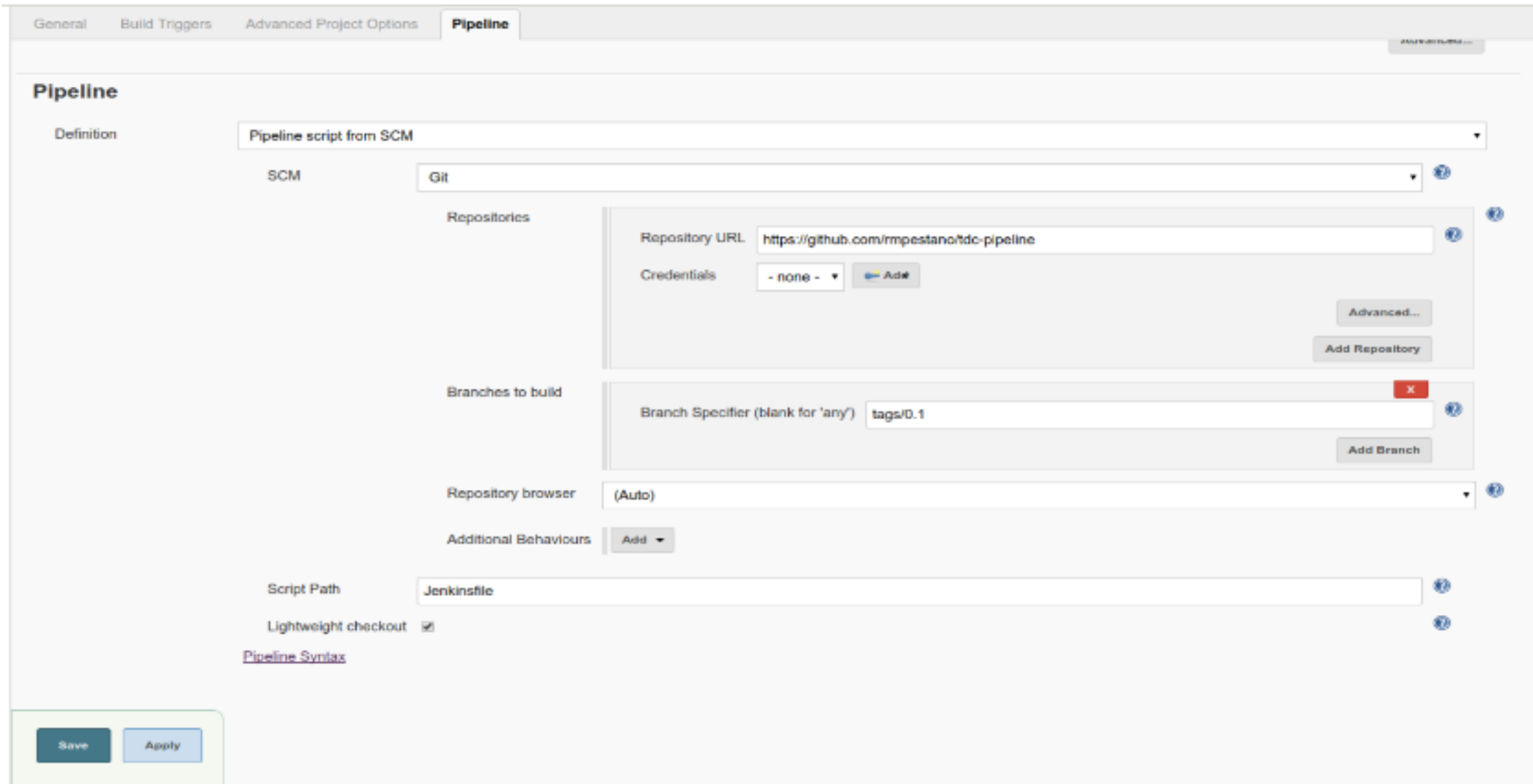
JENKINS 2.X PIPELINE AS CODE

```
pipeline {
  agent any

  stages {
    stage('checkout') {
      steps {
        git 'https://github.com/rmpestano/tdc-pipeline.git'
      }
    }

    stage('build') {
      steps {
        sh 'mvn clean package'
      }
    }
  }
}
```

JENKINS 2.X PIPELINE ON CODE



The screenshot shows the Jenkins Pipeline configuration interface. At the top, there are tabs for 'General', 'Build Triggers', 'Advanced Project Options', and 'Pipeline'. The 'Pipeline' tab is selected. Below the tabs, the 'Pipeline' section is visible. Under 'Definition', a dropdown menu is set to 'Pipeline script from SCM'. Under 'SCM', a dropdown menu is set to 'Git'. Under 'Repositories', the 'Repository URL' is 'https://github.com/rmpestano/tdc-pipeline' and 'Credentials' is set to '- none -'. There are buttons for 'Advanced...', 'Add Repository', and 'Add Branch'. Under 'Branches to build', the 'Branch Specifier (blank for \'any\')' is 'tags/0.1'. There is a button for 'Add Branch'. Under 'Repository browser', a dropdown menu is set to '(Auto)'. Under 'Additional Behaviours', there is an 'Add' button. Under 'Script Path', the text is 'Jenkinsfile'. Under 'Lightweight checkout', there is a checked checkbox. At the bottom left, there are 'Save' and 'Apply' buttons. A link for 'Pipeline Syntax' is also visible.

General Build Triggers Advanced Project Options **Pipeline**

Pipeline

Definition Pipeline script from SCM

SCM Git

Repositories

Repository URL https://github.com/rmpestano/tdc-pipeline

Credentials - none - Add

Advanced... Add Repository

Branches to build

Branch Specifier (blank for 'any') tags/0.1 Add Branch

Repository browser (Auto)

Additional Behaviours Add

Script Path Jenkinsfile

Lightweight checkout ☒

[Pipeline Syntax](#)

Save Apply

Demo v0.1 (<https://github.com/rmpestano/tdc-pipeline/releases/tag/0.1>)

JENKINS 2.X PIPELINE ON CODE

```
pipeline {
  agent any

  stages {

    stage('build') {

      steps {
        sh 'mvn clean package'
      }
    }

    stage('Deploy') {
      steps {
        sh 'docker stop tdc-pipeline || true && docker rm tdc-pi'
        sh 'docker build -t tdc-pipeline .'
```

SONAR

Demo v0.2 (<https://github.com/rmpestano/tdc-pipeline/releases/tag/0.2>)

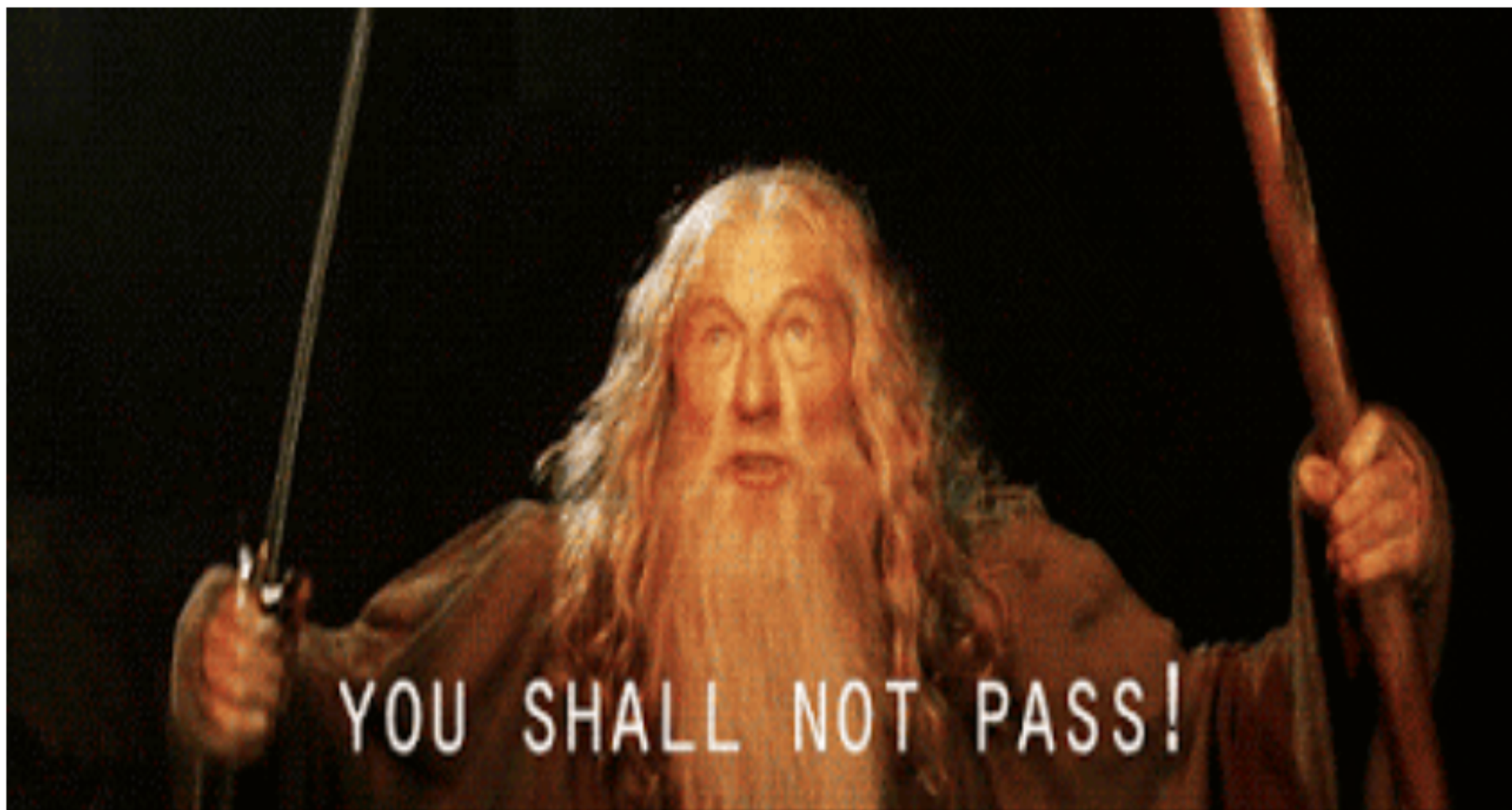


SONAR

```
pipeline {  
  agent any  
  
  stages {  
    stage('build') {  
      steps {  
        sh 'mvn clean package -DskipTests'  
      }  
    }  
  
    stage('unit-tests') {  
      steps {  
        sh 'mvn test -Pcoverage'  
      }  
    }  
  }  
}
```

QUALITY GATE

Demo v0.3 (<https://github.com/rmpestandano/tdc-pipeline/releases/tag/0.3>)



QUALITY GATE

```
pipeline {  
  agent any  
  
  stages {  
    stage('build') {  
      steps {  
        sh 'mvn clean package -DskipTests'  
      }  
    }  
  
    stage('unit-tests') {  
      steps {  
        sh 'mvn test -Pcoverage'  
      }  
    }  
  }  
}
```

POST ACTIONS

Demo v0.4 (<https://github.com/rmpestano/tdc-pipeline/releases/tag/0.4>)

```
pipeline {
  agent any

  //stages

  post {
    always {
      sendNotification(currentBuild.result)
    }

    success {
      echo 'Build was a success'
    }

    failure {
      echo 'Build failure'
    }
  }
}
```

PIPELINE SHARED LIBRARIES

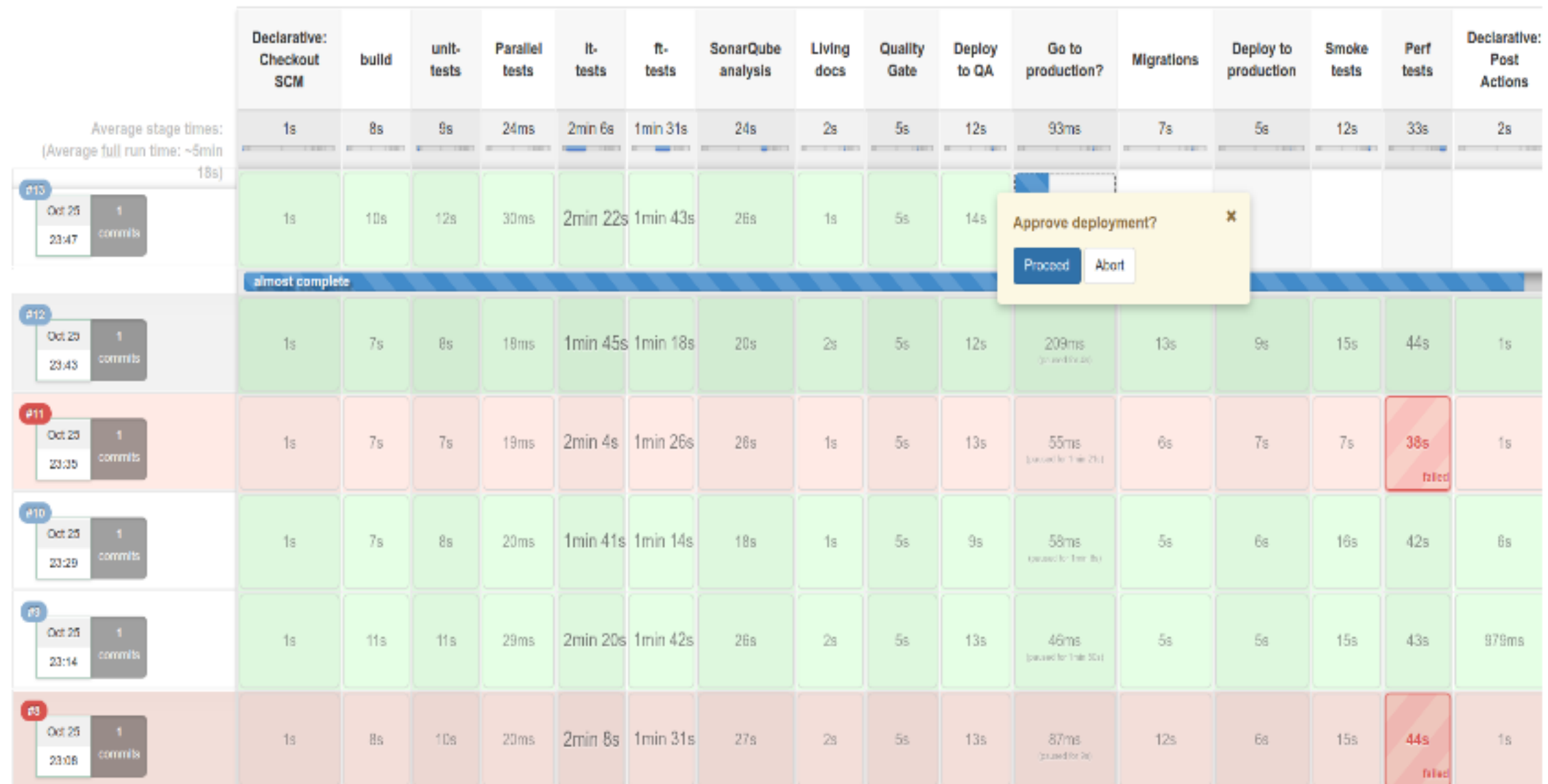
Enables reuse of pipeline sections (even entire stages) among projects

<https://github.com/rmpestano/tdc-pipeline#shared-library>

TDC PIPELINE FINAL


Jenkins > tdc-pipeline > Full Stage View

tdc-pipeline - Stage View



VIDEO

Java EE Pipeline as code with Jenkins, Docker and Sonar



The screenshot displays the Jenkins CI/CD pipeline interface for a Java EE Pipeline. The interface shows a build status of 'Success' with a green checkmark. Below this, there are several metrics: 'Builds' (1), 'Tests' (1), 'Coverage' (52.5%), and 'Issues' (0). The 'Coverage' section shows a bar chart with a value of 52.5% and a target of 42. The 'Issues' section shows a bar chart with a value of 0.0% and a target of 0. The interface also includes a 'Summary' section on the right with various links and a 'Log' section at the bottom showing the pipeline script.

```
1 pipeline {
2   agent {
3     docker {
4       image 'java:8-jdk-alpine'
5     }
6   }
7   stages {
8     stage('SonarQube analysis') {
9       steps {
10         checkout($REPO)
11         withSonarQubeEnv('sonar') {
12           sh 'mvn sonar:sonar'
13         }
14       }
15     }
16     stage('Build & Test') {
17       steps {
18         checkout($REPO)
19         mvn clean install -DskipTests
20       }
21     }
22     stage('Deploy') {
23       steps {
24         checkout($REPO)
25         mvn clean install
26       }
27     }
28   }
29 }
```

PERGUNTAS?



REFERENCES

- <https://github.com/rmpestano/tdc-pipeline/>
- <https://jenkins.io/doc/book/pipeline/syntax/>
- <https://jenkins.io/blog/2017/02/15/declarative-notifications/>
- <https://jenkins.io/doc/book/pipeline/shared-libraries/>
- <https://jenkins.io/blog/2017/02/07/declarative-maven-project/>
- <https://virtualjug.com/pipeline-as-code-building-continuous-delivery-pipelines-with-jenkins-2/>

Slides: <https://rmpestano.github.io/talks/slides/javaee-pipeline/index-en.html>



VALEU

GALERA!