Dr. Michael Eichberg
Software Engineering
Department of Computer Science
Technische Universität Darmstadt

Software Engineering

Building Software

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Non-trivial Software is generally Build using Build Automation Systems.

- The goal of a Build Automation System is to **fully automate all steps** required to build the product given the source artifacts of the project.

The result of the build should always be the same - independent of the developer's local configuration.

"We want stable builds."

The Build Automation Systems is responsible for automatically carrying out all steps necessary to build the product.

- A Build Automation typically executes the following tasks:
  - Formatting the source code
  - Code Generation
  - Source Code Compilation
  - [if necessary] Linking Code/Packaging Code
  - Running the tests
  - Running static analysis tools
  - Deployment to the test system/production system(s)
  - Creating and publishing documentation, release notes, web pages, …

Historically

Software is Build using Build Automation Systems.

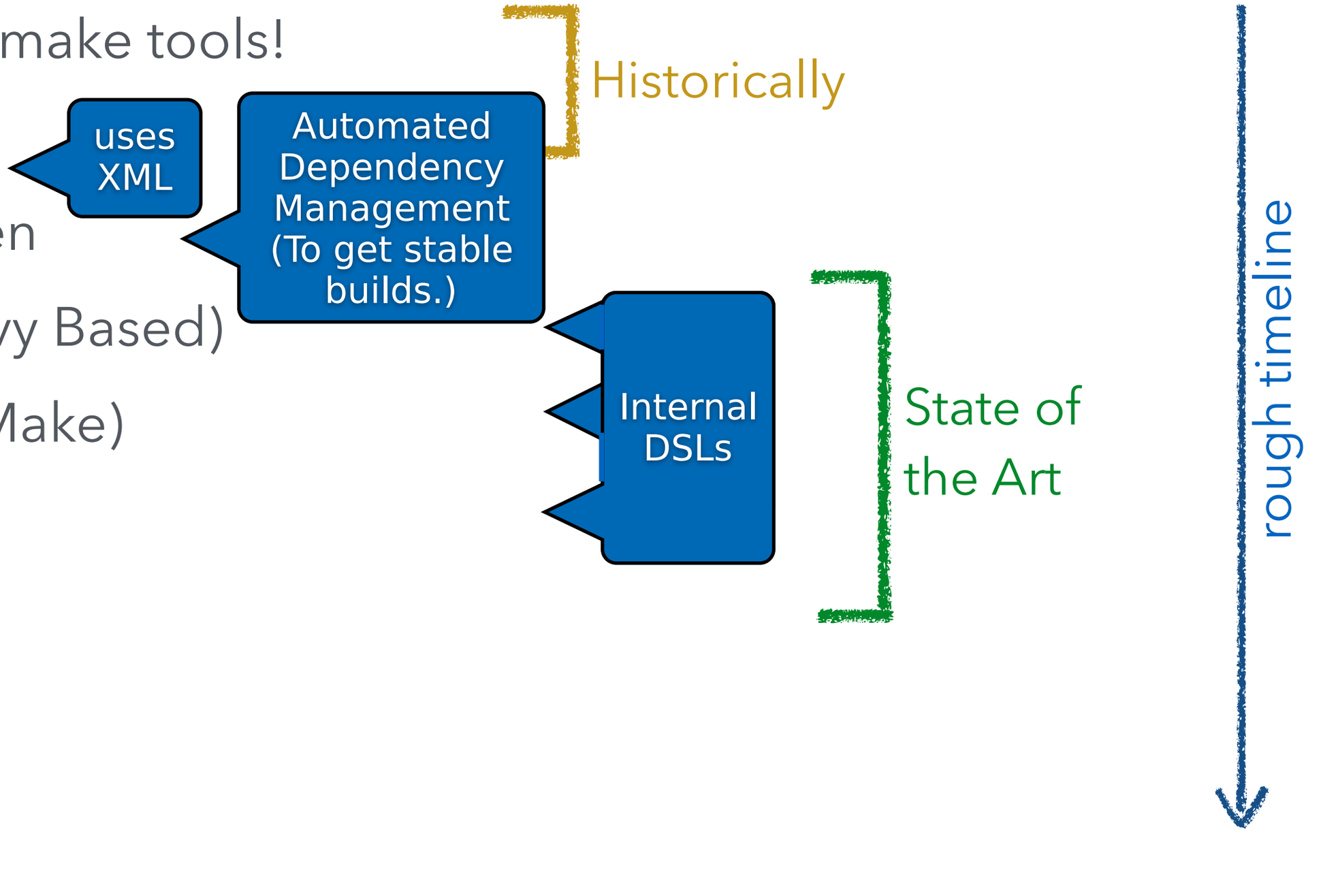
- Given a Build Automation System, the product can be built:
  - On-Demand
    (e.g., by a developer)
  - Scheduled by a build server
    (e.g., every night)
  - Triggered
    (e.g., on every commit to a version control system)

Historically

State of the Art

Some Examples of (Open-Source) Tools to Automate Builds

- The family of make tools!
- Apache Ant
- Apache Maven
- gradle (Groovy Based)
- RAKE (Ruby Make)
- sbt
- …



```
import AssemblyKeys._

name := "BugPicker"

version := "1.1.0"

scalaVersion := "2.11.4"

scalacOptions in (Compile, doc) := Seq("-deprecation", "-feature", "-un

scalacOptions in (Compile, doc) ++= Opts.doc.title("OPAL - BugPicker")

libraryDependencies += "org.scalafx"  %% "scalafx"   % "1.0.0-R8"

jfxSettings

JFX.addJfxrtToClasspath := true

JFX.mainClass := Some("org.opalj.bugpicker.BugPicker")

assemblySettings

jarName in assembly := "bugpicker-" + version.value + ".jar"

test in assembly := {}

mainClass in assembly := Some("org.opalj.bugpicker.BugPicker")

resourceGenerators in Compile <+= Def.task {
 val versionFile = (baseDirectory in Compile).value / "target" / "scala-2.11" / "classes" / "org" /
"opalj" / "bugpicker" / "version.txt"
 versionFile.getParentFile.mkdirs()
 IO.write(versionFile, (version in Compile).va
 Seq(versionFile)
}
```

Version Information

Compiler Settings

Project Dependencies

Project Settings

Deployment information

Generation of other Artifacts

Easily hundreds of lines for larger projects.

Continuous Integration

- Continuous integration basically just means that the developer's working copies are synchronized with a shared mainline several times a day.
  It was first named and proposed by Grady Booch.
- The goal is to avoid integration issues.
- CI is in particular useful in combination with automated unit tests.
- In practice a special build server is used.
  (e.g., Hudson/Jenkins)

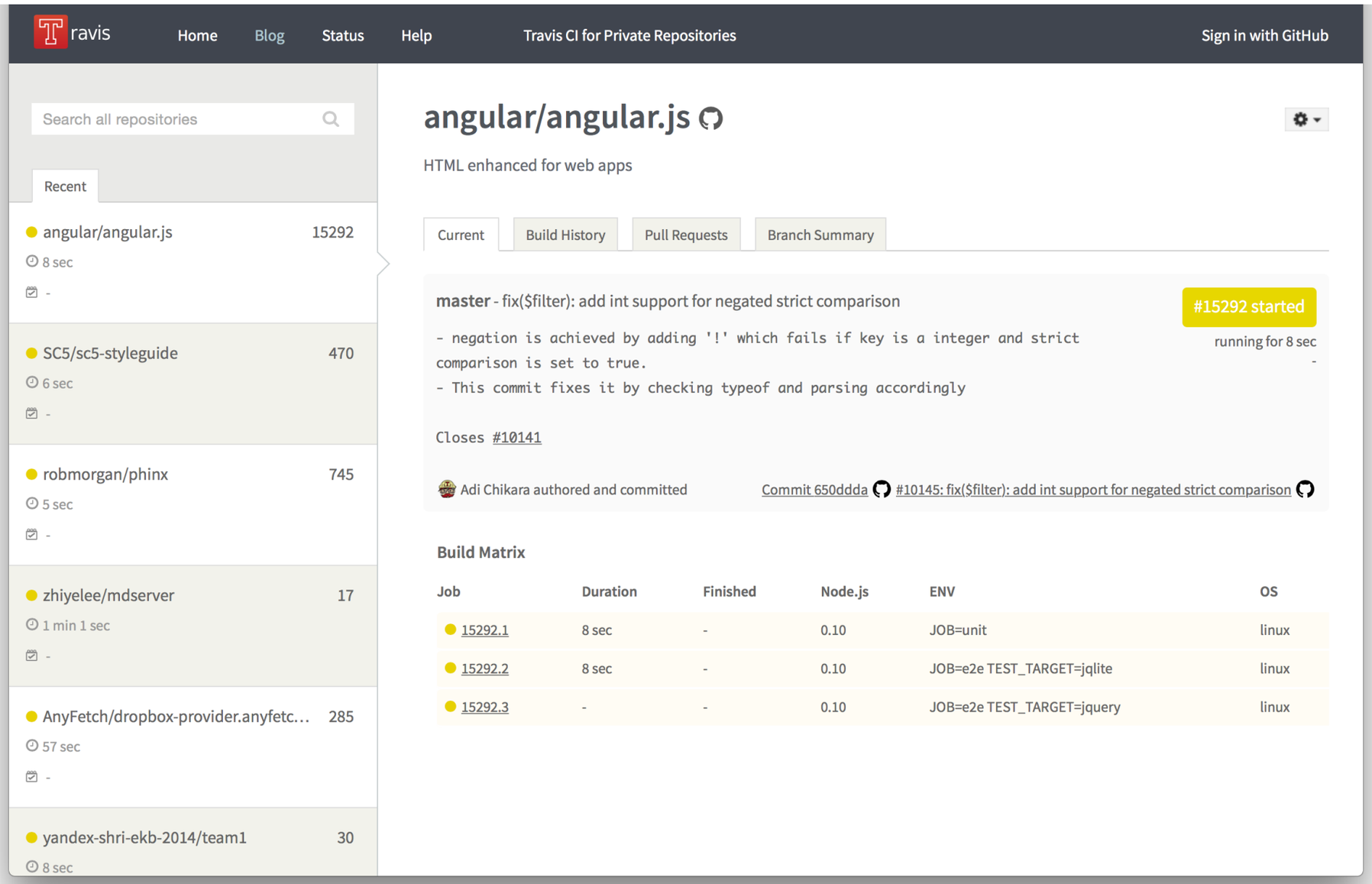
Continuous Integration - Best Practices

- Maintain a code repository
- Automate the build
- Make the build self-testing
- Everyone commits to the baseline every day
- Every commit (to baseline) should be built
  One commit - one feature; no “Mega-commits”
- Keep the build fast
- Test in a clone of the production environment
- Make it easy to get the latest deliverables
- Everyone can see the results of the latest build
- Automate deployment

must haves
standard

Travis CI

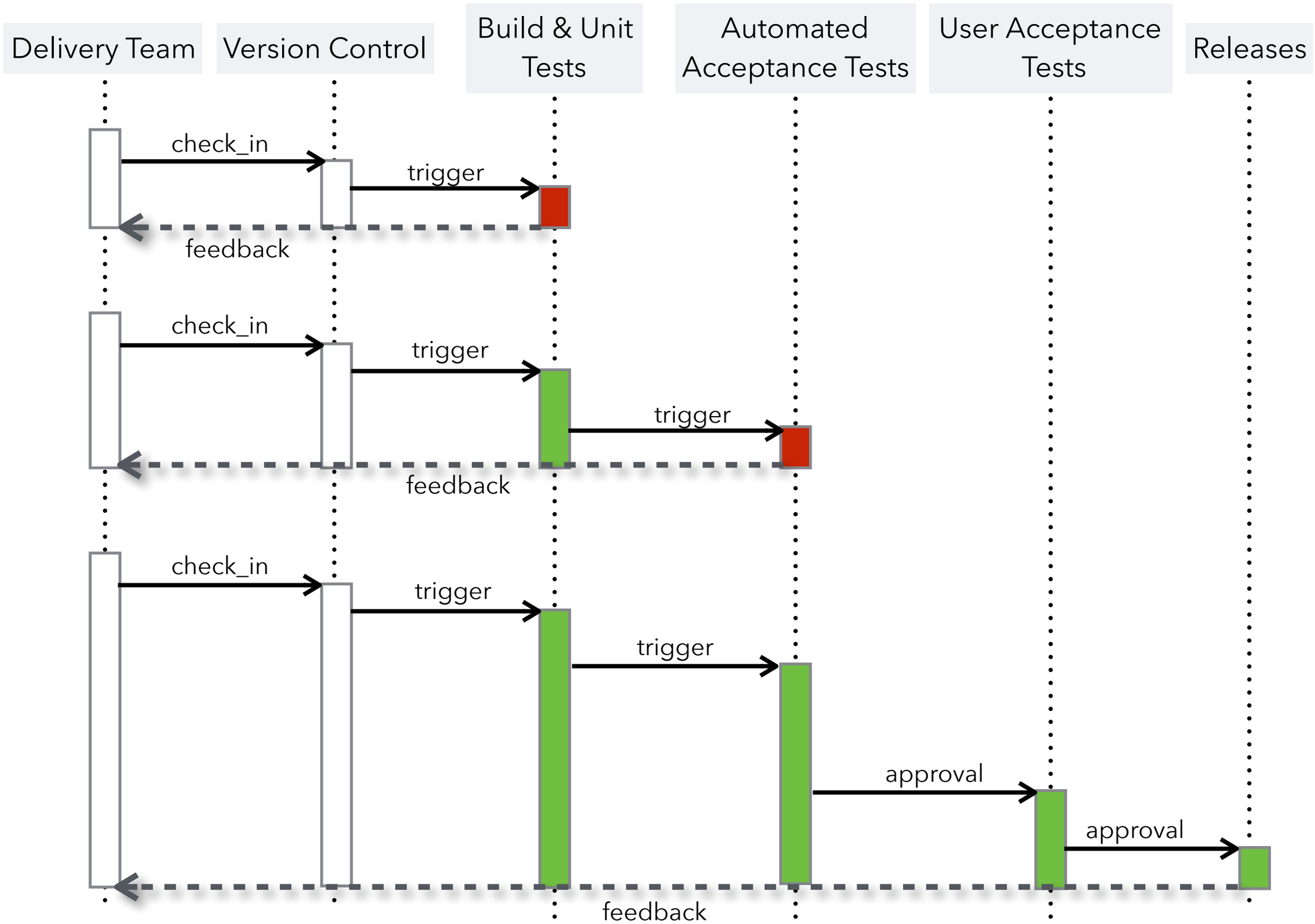
- A hosted continuous integration service for open source and private projects.



Continuous Delivery

- *Always be able to put a product into production (The evolution of continuous integration.)*
- Practices
  - Unit/Acceptance-tests
  - Code coverage and static analysis
  - Deployment to integration environment
  - Integration tests
  - Deployments to performance test environment
  - Performance tests
  - Alerts, reports and release notes sent out
  - Deployment to release repository

Continuous Delivery



A PRINCIPLE OF SOFTWARE DELIVERY: **BUILD QUALITY IN!**

BUSINESS

IT

DEV. + ACCEPT.

QUEUE TO PRODUCTION

I CAN SHIP MY IDEAS WHENEVER I LIKE!

BUSINESS

GO

STOP

CUSTOMER

CONTINUOUS DELIVERY

BY JEZ HUMBLE & DAVID FARLEY

DONE MEANS RELEASED

A CLOSER LOOK-

COMMIT STAGE

- ☑ CREATING EXECUTABLE CODE MUST WORK. VERIFIES THAT THE SYNTAX OF YOUR SOURCE CODE IS VALID
- ☑ UNIT TEST PASS
- ☑ FULFILL CERTAIN QUALITY CRITERIA SUCH AS TEST COVERAGE AND OTHER TECHNOLOGY-SPECIFIC METRICS

KEY *pattern* — DEPLOYMENT PIPELINE

EXAMPLE

COMMIT STAGE	AUTOMATED ACCEPTANCE TESTING	AUTOMATED CAPACITY TESTING	MANUAL TESTING	RELEASE
BUILD UNIT TEST			SHOW CASES EXPLORATORY TESTING	
✓ PASS	✓ PASS	✓ PASS	✓ RUN	

FAST → SLOW

SHOWSTOPPERS → NOT NECESSARY SHOWSTOPPERS

ENVIRONMENT NEUTRAL → PRODUCTION LIKE ENVIRONMENT

FEED-BACK

CHANGE

CHANGE IN
- EXEXUTABLE CODE
- CONFIGURATION
- HOST ENVIRONMENT
- DATA

CREATE NEW INSTANCE OF PIPELINE

CHANGE1 → PIPELINE 1
CHANGE 2 → PIPELINE 2
CHANGE3 → PIPELINE 3

- ANY CHANGE IS A TRIGGER
- FAST
- ACT ON IT

BENEFITS

EMPOWERED - IN CONTROL
LOW STRESS - SMALL RELEASES

REDUCING ERRORS
- CONFIG MGT.
- VERSION CONTROL

DEPLOYMENT FLEXIBILITY - EASY TO START APPLICATION IN NEW ENVIRONMENT

PRACTICE MAKES *Perfekt*

SEEMS LIKE THE AUTHORS CAN'T STRESS IT ENOUGH. IT'S EVERYWHERE THROUGHOUT THIS BOOK.

VERSION CONTROL

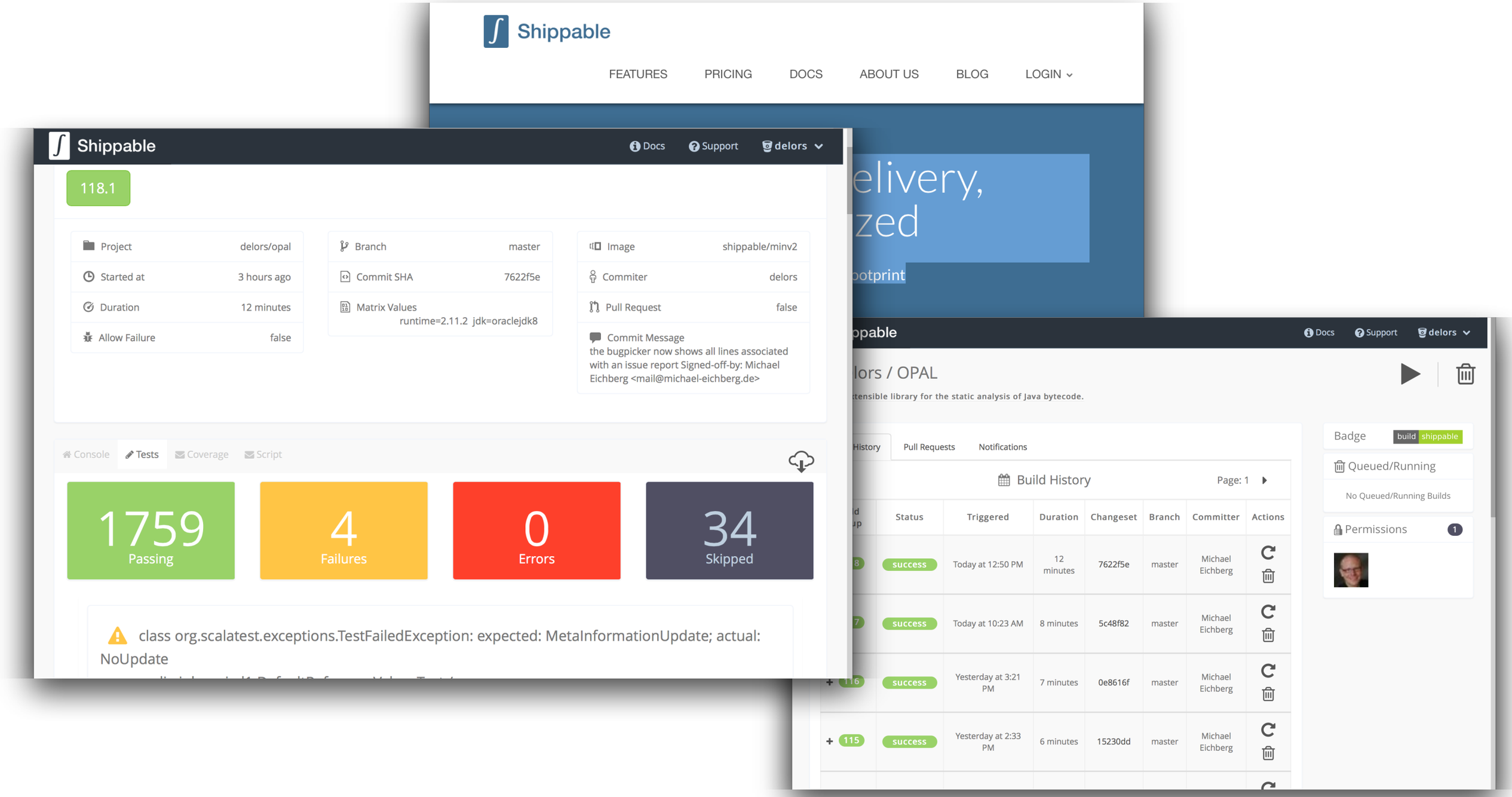
AUTOMATE ALMOST EVERYTHING

> ENCOURAGING GREATER COLLABORATION BETWEEN EVERYONE INVOLVED IN SOTFWARE DELIVERY IN ORDER TO RELEASE VALUABLE SOFTWARE FASTER AND MORE RELIABLY.

If it hurts, do it more frequently

Nhan Ngo

Cloud Services for Continuous Delivery



Continuous Deployment

- Automatically **deploy the product into production** whenever it passes QA.
  (The logical next step after Continuous Delivery)
- The release schedule is in the hands of the IT department
  (With Continuous Delivery the release schedule is in the hands of the business.)

Attention: Sometimes the term “Continuous Deployment” is also used if you are able to continuously deploy to the test system.

Summary

The goal of this lecture is to enable you to systematically carry out small(er) software projects that produce quality software.

- Projects are build using build tools
- A build script takes care of all steps necessary to build the project

  (In case of an application, building means creating a runnable application.)

- The goal of this lecture is to enable you to systematically carry out small(er) commercial or open-source projects.