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, ...
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)
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
- ...

Historically

Automated Dependency Management (To get stable builds.)

Internal DSLs

uses XML

State of the Art

rough timeline
import AssemblyKeys._

name := "BugPicker"

version := "1.1.0"

scalaVersion := "2.11.4"

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

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).value)
  Seq(versionFile)
}
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
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

© http://continuousdelivery.com/2010/02/continuous-delivery/ | 11
A PRINCIPLE OF SOFTWARE DELIVERY: BUILD QUALITY IN!

CONTINUOUS DELIVERY

BY JEZ HUMBLE & DAVID PARLEY

A CLOSER LOOK

COMMITS

CREATE EXCUTABLE 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

COMMIT STAGE

BUILD

UNIT TEST

AUTOMATED

ACCEPTANCE

TESTING.

AUTOMATED

CAPACITY

TESTING.

MANUAL TESTING

SHOWCASES EXPLORATORY TESTING

RELEASE

DONE MEANS RELEASED

EXAMPLE

CHANGE

IN

EXECUTABLE CODE

CONFIGURATION

HOST ENVIRONMENT

DATA

CREATE NEW INSTANCE OF PIPELINE

CHANGE 1

CHANGE 2

CHANGE 3

PIPELINE 1

PIPELINE 2

PIPELINE 3

FEEDBACK

ANY CHANGE IS A TRIGGER • FAST • ACT ON IT

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

“

If it hurts, do it more frequently

“

BENEFITS

EMPOWERED • IN CONTROL
LOW STRESS • SMALL RELEASES

REDUCING ERRORS • CONFIG MGMT. • VERSION CONTROL

PRACTICE MAKES PERFECT

AUTOMATE ALMOST EVERYTHING

SEEMS LIKE THE/authors CAN’T STRESS IT ENOUGH. IT’S EVERYWHERE THROUGHOUT THIS BOOK...

VERSION CONTROL

If it hurts, do it more frequently.
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.
The goal of this lecture is to enable you to systematically carry out small(er) software projects that produce quality software.

- Projects are built 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.