Software Engineering Essentials

Build Management

Bernd Bruegge, Stephan Krusche, Andreas Seitz, Jan Knobloch
Chair for Applied Software Engineering — Faculty of Informatics

Learning Goals

1) Explain the idea of continuous integration
2) Describe the benefits of regression testing

Typical situations in projects

"On my computer, it compiled!"

Developer

"You don't like the new design? Why didn't you tell me earlier? I wasted so much time on that!"

"It's been a long time since the last release …"

"I don't like this design!"

Manager

Customer

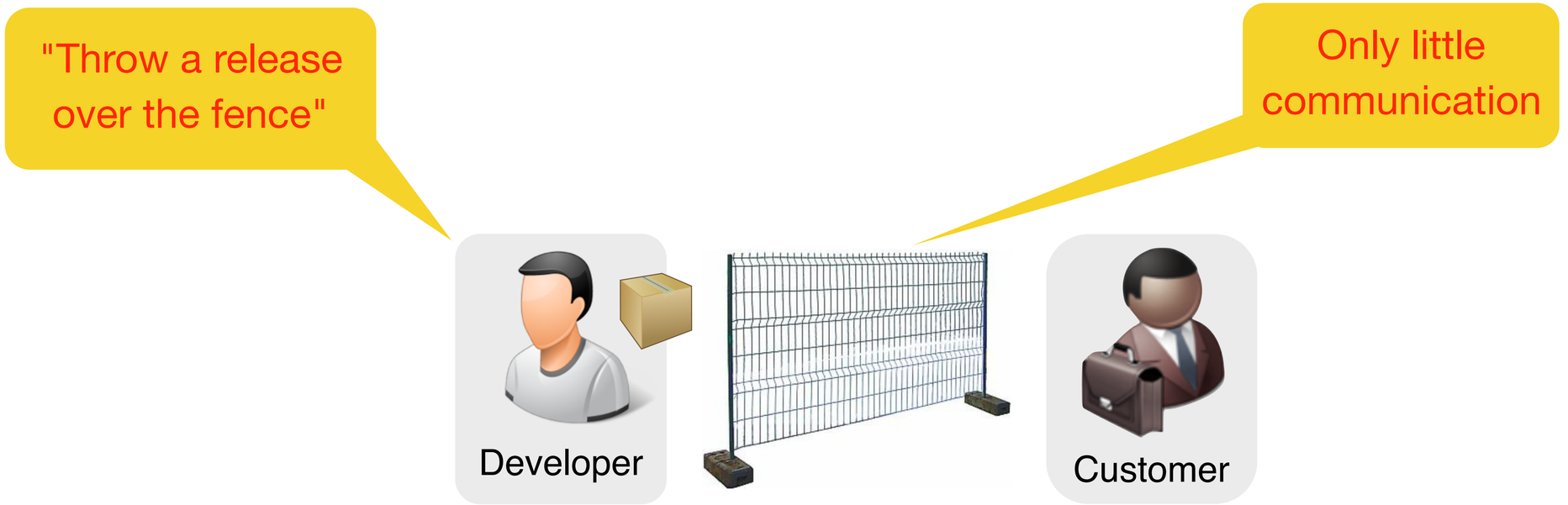
"What's the current status?"

"The software is not working!"

How it is still often done?

Delivery only after the implementation was finished

- No releases during development
- No feedback from customers and users during development



development environment != target environment

Objective: Release source code to devices in the target environment

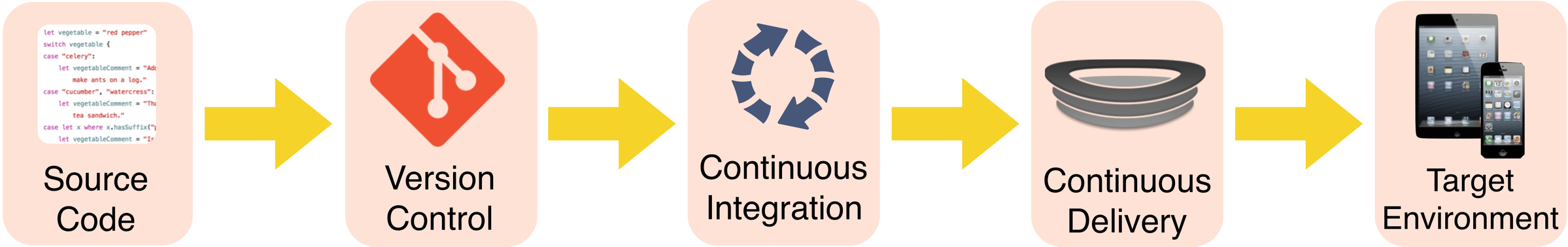
Source Code

Target Environment

Developer

Customer

Build and release management overview



Developer

Customer

Review: software configuration management activities

1) Configuration item identification
- Modeling the system as set of evolving components

2) Promotion management
- Creation of versions for other developers

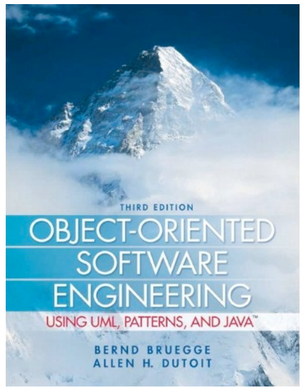
3) Build and release management
- Creation of versions for customers and end users

4) Change management
- Handling, approval & tracking of change requests

Covered in this course

5) Branch management
- Management of concurrent development

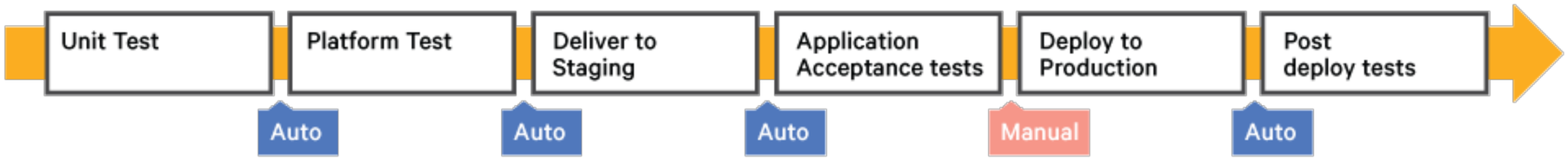
6) Variant management
- Management of coexisting versions



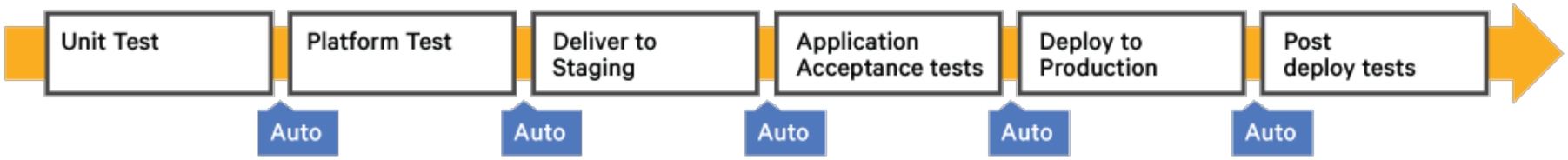
Bruegge, Dutoit: Object-Oriented Software Engineering Using UML, Patterns, and Java (Chapter 13)

Terminology

- Continuous integration: technique where members of a team integrate their work frequently: usually each person integrates at least daily, leading to multiple integrations per day.
- Continuous delivery: approach in which teams keep producing valuable software in short cycles and ensure that the software can be reliably released at any time.

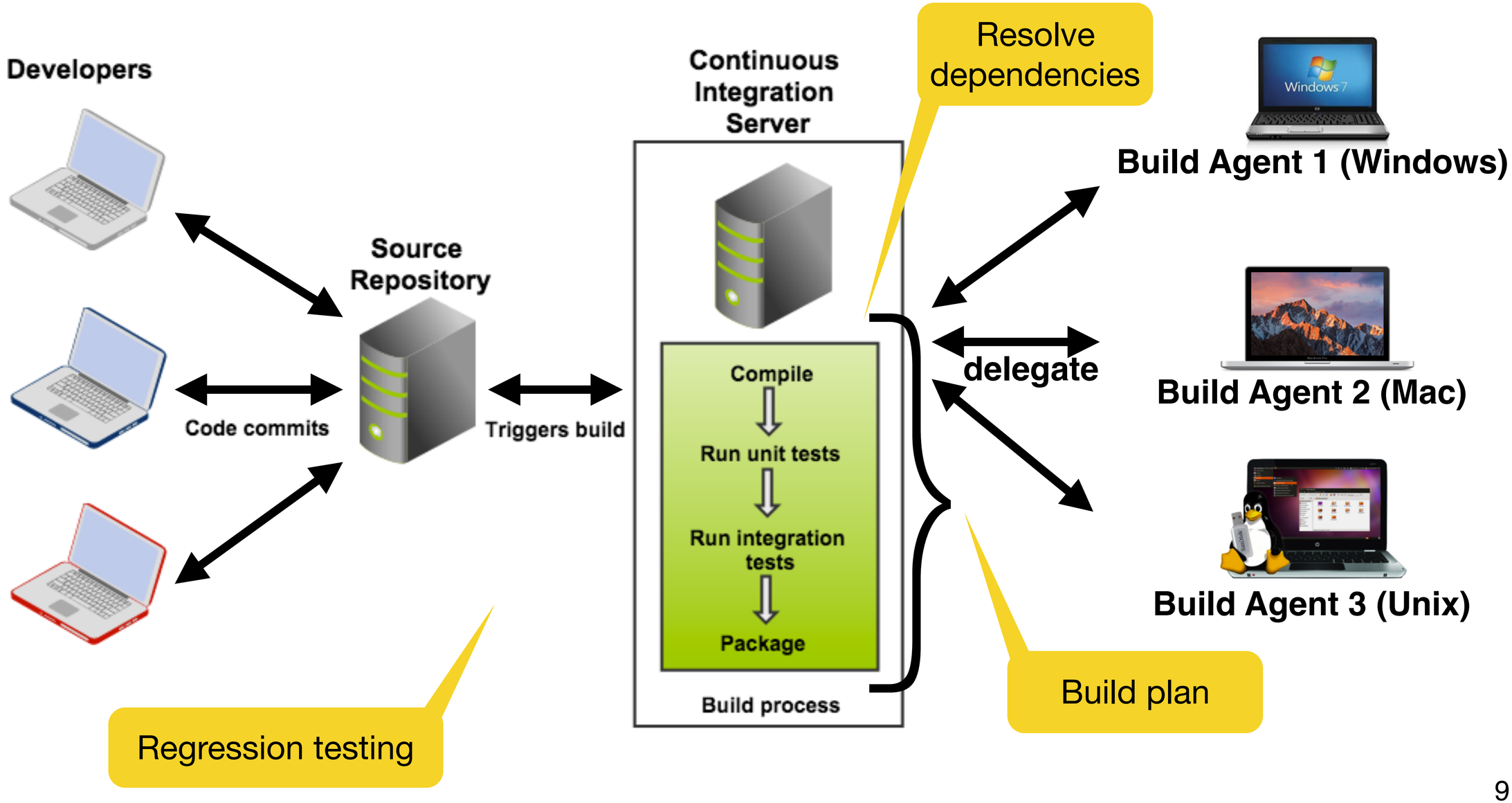


- Continuous deployment: every change that passes the automated tests is deployed automatically (usually incrementally)



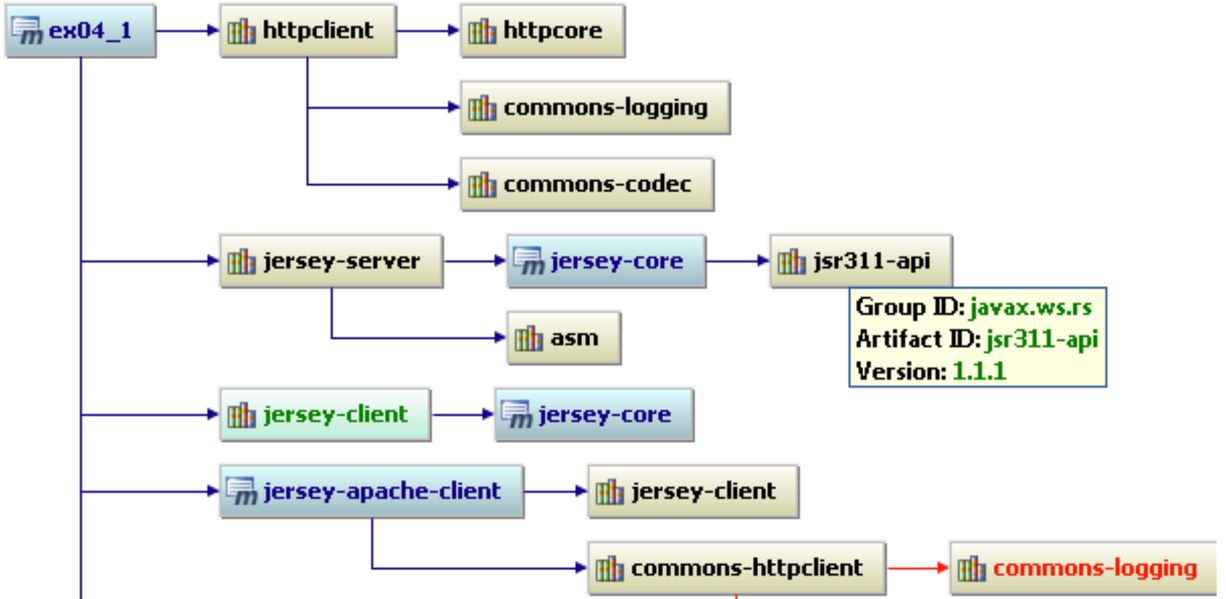
Source: https://puppet.com/blog/continuous-delivery-vs-continuous-deployment-what-s-diff

Build management with continuous integration



Maven™ : build and dependency management tool

- Open source build and dependency management tool for Java projects
- Stores libraries, frameworks and plugins in a central repository: MavenCentral
  - Easy reuse of existing components
- **POM** = **P**roject **O**bject **M**odel (main artifact and configuration file: pom.xml)
- Support for multiple build lifecycle phases, e.g. compile, test, package, install, deploy
  - More information: https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html
- Dependency example:



Example pom.xml

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>de.tum.in.www1.seecx.university</groupId>
  <artifactId>UniversityApp</artifactId>
  <packaging>jar</packaging>
  <version>1.0</version>
  <name>UniversityApp</name>
  <dependencies><dependency>
      <groupId>com.mashape.unirest</groupId>
      <artifactId>unirest-java</artifactId>
      <version>1.4.9</version>
  </dependency></dependencies>
  <build>
    <sourceDirectory>src</sourceDirectory>
    <plugins><plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.3</version>
      </plugin>
      <plugin>
        <artifactId>maven-jar-plugin</artifactId>
        <version>3.0.2</version>
        <configuration><archive><manifest>
          <mainClass>de.tum.in.www1.seecx.UniversityApp</mainClass>
        </manifest></archive></configuration>
    </plugin></plugins></build>
</project>
```

Packaging format

Dependency to external framework

Build and package instructions

Typical commands

- mvn clean
- mvn compile
- mvn test
- mvn package
- mvn deploy

Regression testing

Goal: verify that software previously developed and tested still performs correctly even after it was changed or interfaced with other software

\+ **Benefit:** finds errors in the existing source code immediately after a change is introduced

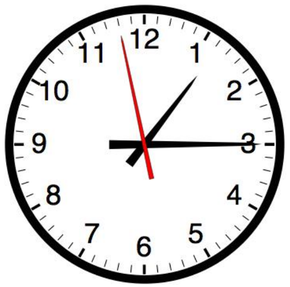
\- **Drawback:** can be costly to execute a large test suite after each change

Techniques:

- Retest all
- Regression test selection (e.g. using dependency analysis)
- Test case prioritization

When to execute the selected test cases?

- After each change
- Nightly
- Weekly



Software Engineering Essentials

Build Management

Bernd Bruegge, Stephan Krusche, Andreas Seitz, Jan Knobloch
Chair for Applied Software Engineering — Faculty of Informatics