POM Class 13

**Requirements** **for build management**

Large and distributed software projects need to provide a development infrastructure with an integrated build management that supports:

* Regular builds from the master directory
* Automated execution of tests
* E-mail notification
* Determination of code metrics
* Automated publishing of the applications and test results

**Tools** for build management: Unix’s Make, Ant, Maven

The transition from **source code to the executable application** contains many mechanical activities. Executing these steps manually is time consuming and the chance of introducing failures is high -> Automating these steps

**Reasons for continuous integration**

* The later integration occurs in a project, the bigger is the risk that unexpected failures occur
* The higher the complexity of the software system, the more difficult is the integration of its components
* Continuous integration addresses these risks by integrating and testing as early and frequently as possible

**Begriffe**

* **Failure**: Any deviation of the observed behavior from the specified behavior (crash)
* **Error**: The system is in a state such that further processing by the system can lead to a failure
* **Fault**: The mechanical or algorithmic cause of an error (bug)
* **Validation**: Activity of checking for deviations between the observed behavior of a system and its specified behaviour

**How to deal with faults**

**Fault avoidance**

* Use methodology to reduce complexity
* Use software configuration management to prevent inconsistency
* Apply verification to prevent algorithmic faults
* Use reviews to identify faults already in the design

**Fault** **detection**

* Testing: Provoke failures in a planned way
* Debugging: Find and remove faults
* Monitoring: Deliver information about state and unusual behavior => Used during debugging

**Fault** **tolerance**

* Exception handling, modular redundancy

**Types of testing**

**Unit** **Testing**

Individual components (class or subsystem) are tested. Confirms that the component is correct

**Integration** **Testing**

Groups of subsystems and eventually the entire system are tested. Tests the interfaces of the subsystem

**System** **Testing**

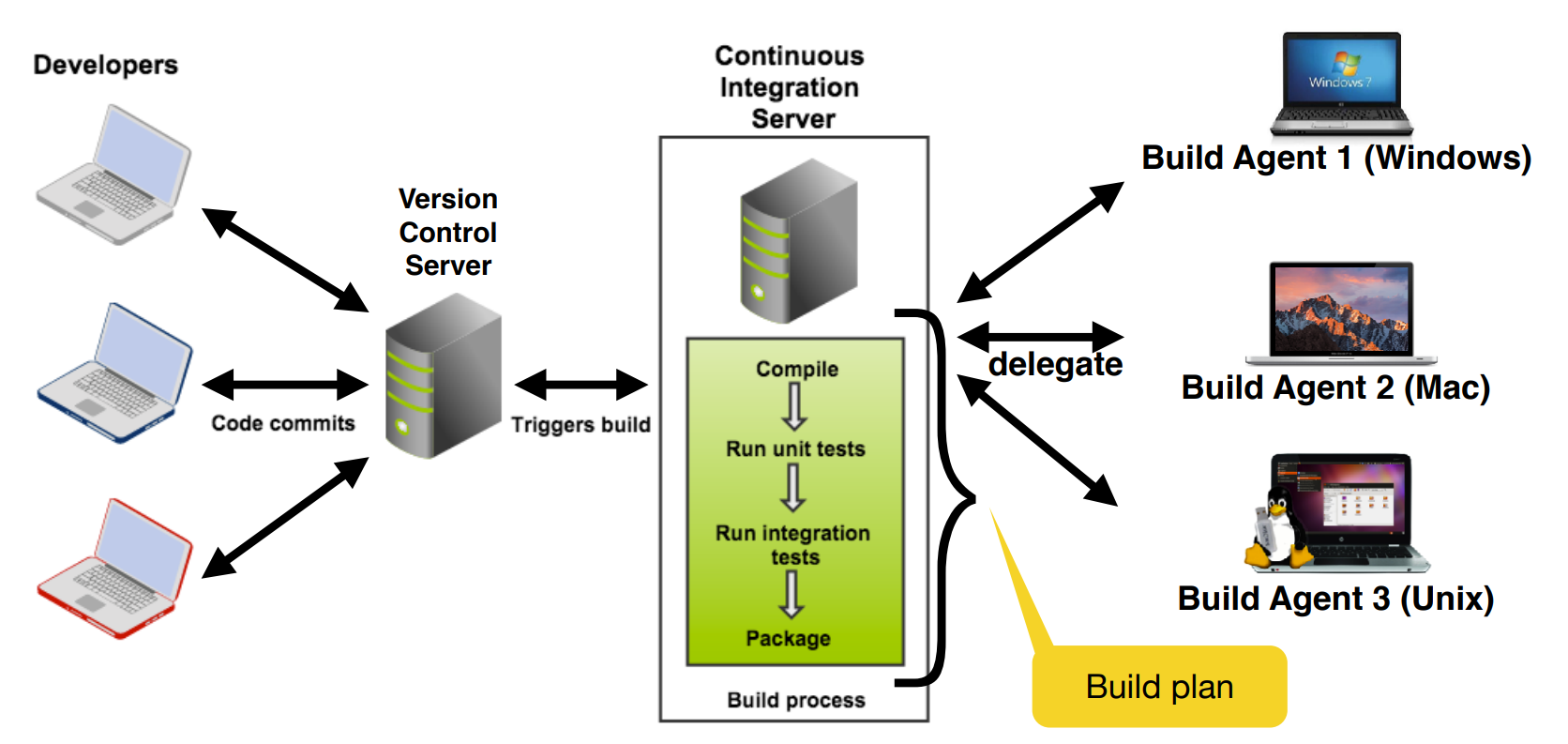
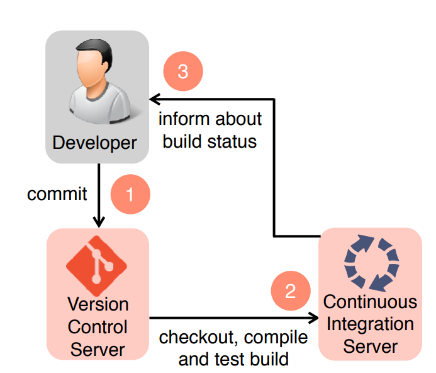
The entire system is tested. Checks if the system meets the requirements

**Acceptance** **Testing**

Evaluates the system delivered by developers (Carried out by the client). Checks if the system meets the requirements and if its ready to use

* **Alpha test** 
  + Client uses the software at the developer’s environment
  + Software used in a controlled setting, with the developer always ready to fix bugs
* **Beta test** 
  + Conducted at client’s environment
  + Software gets executed in a realistic target environment

**Continuous Integration:** A software development technique where members of a team integrate their work frequently. Usually each person integrates at least daily, leading to multiple integrations per day. Each integration is verified by an automated build which includes the execution of tests



**Advantages** of continuous integration

* There is always an executable version of the system
* Developers and managers have a good overview of the project status
* Automatic regression testing

**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 very costly to execute a large test suite after each change