



Universidad de Oviedo



# Software taxonomies

## Patterns, styles, tactics,...



SOFTWARE  
ARCHITECTURE

Course 2018/2019

Jose E. Labra Gayo

# Software taxonomies

Construction & Maintenance

Configuration management

Modularity

Decomposition at building time

Runtime

Components and connectors

Integration

Allocation

Packaging, distribution, deployment

Business and enterprise environment



Universidad de Oviedo



# Software construction & maintenance



SOFTWARE  
ARCHITECTURE

Course 2018/2019

Jose E. Labra Gayo

# Software construction & maintenance

## Configuration management



# Software: product or service

## Software as a Product (SaaP):

Software deliverable

Commercial model: software is sold to clients

Software distributed or downloaded

Example: Microsoft Office

## Software as a Service (SaaS):

Software deployed

Commercial model: clients subscribe to it

Software usually available at some URL

Example: Google docs

# Software configuration management

Managing the evolution of software

Manages all aspects of software construction

Especially, how software evolves and changes

Aspects:

Identifying baselines and configuration items

Baseline: A work product subject to management

It contains configuration items: documents, code files, etc...

Configuration control & auditing

Version control systems

Building management and automation

Teamwork

Defect and issues tracking

# Software construction

Overview of methodologies

Traditional, iterative, agile

Construction tools

Languages, tools, etc.



# Incremental piecemeal

Development by need

Codification without following the architecture

Throw-away software

Budget constraints

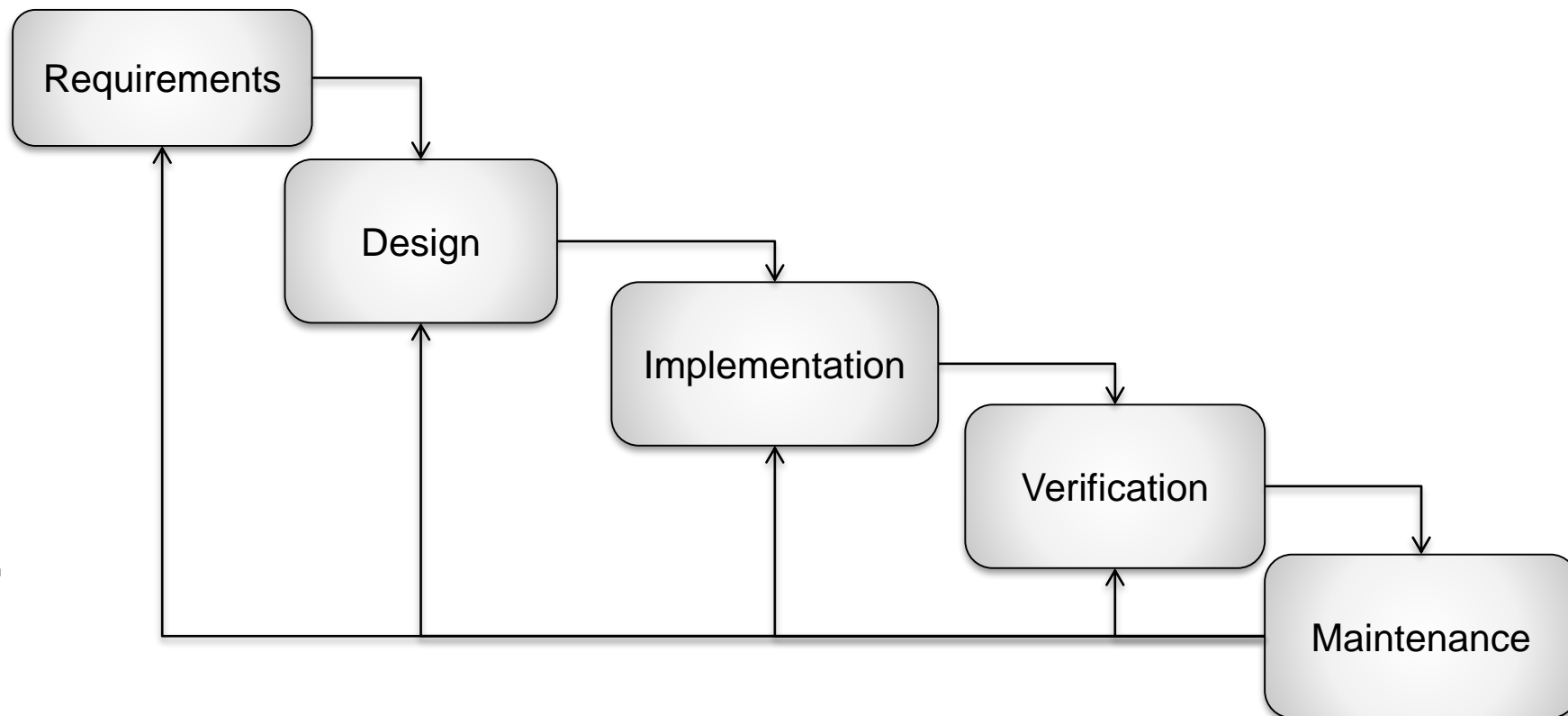




# Waterfall

Software development life cycle (SDLC)

Waterfall model identified as antipattern in 1970s



# V Model

Detail Level

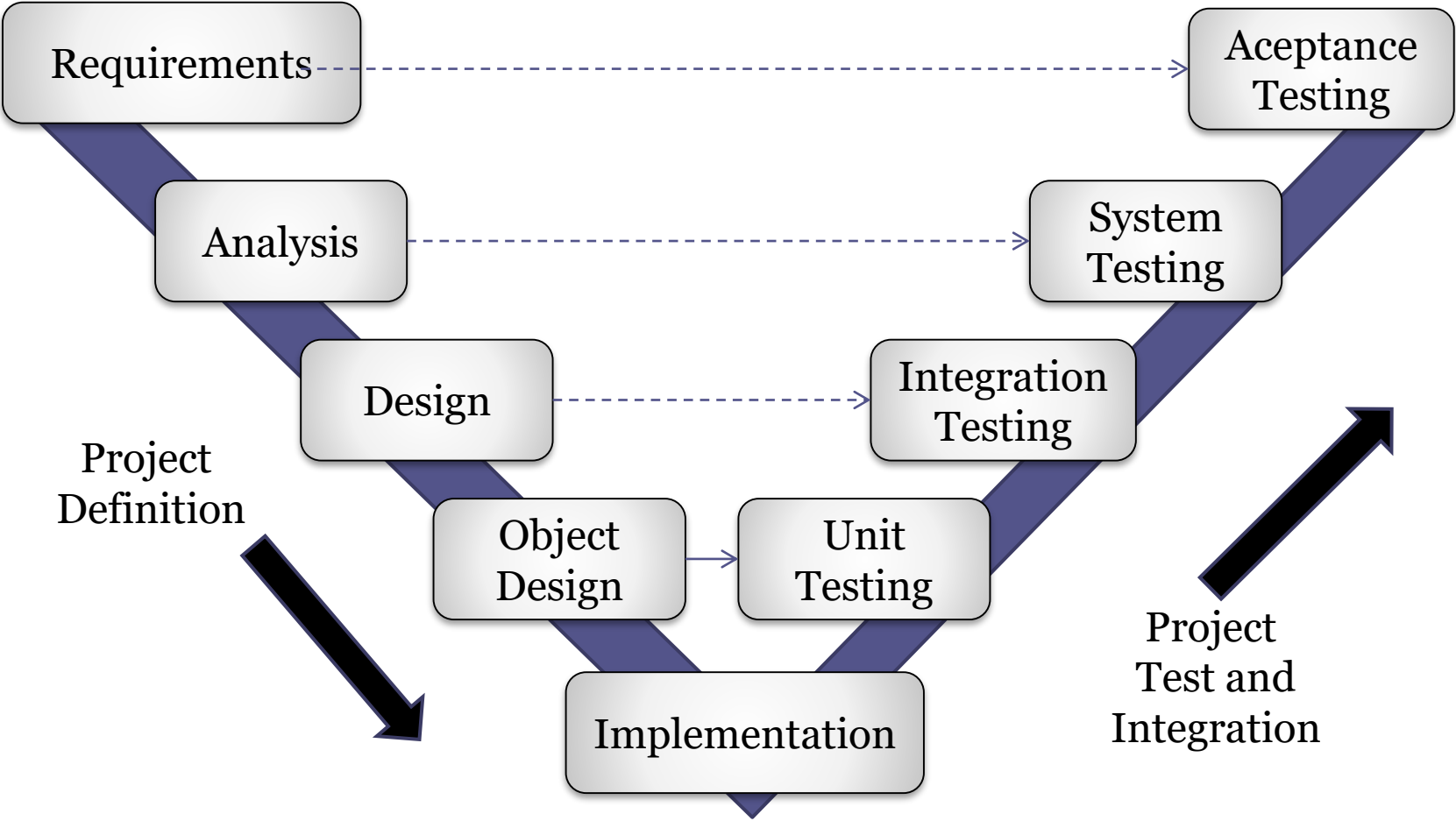
Low

High

Project Definition

Project Test and Integration

Time



# Big Design Up Front

Anti-pattern of traditional models

Too much documentation that nobody reads

Documentation different from developed system

Architecture degradation

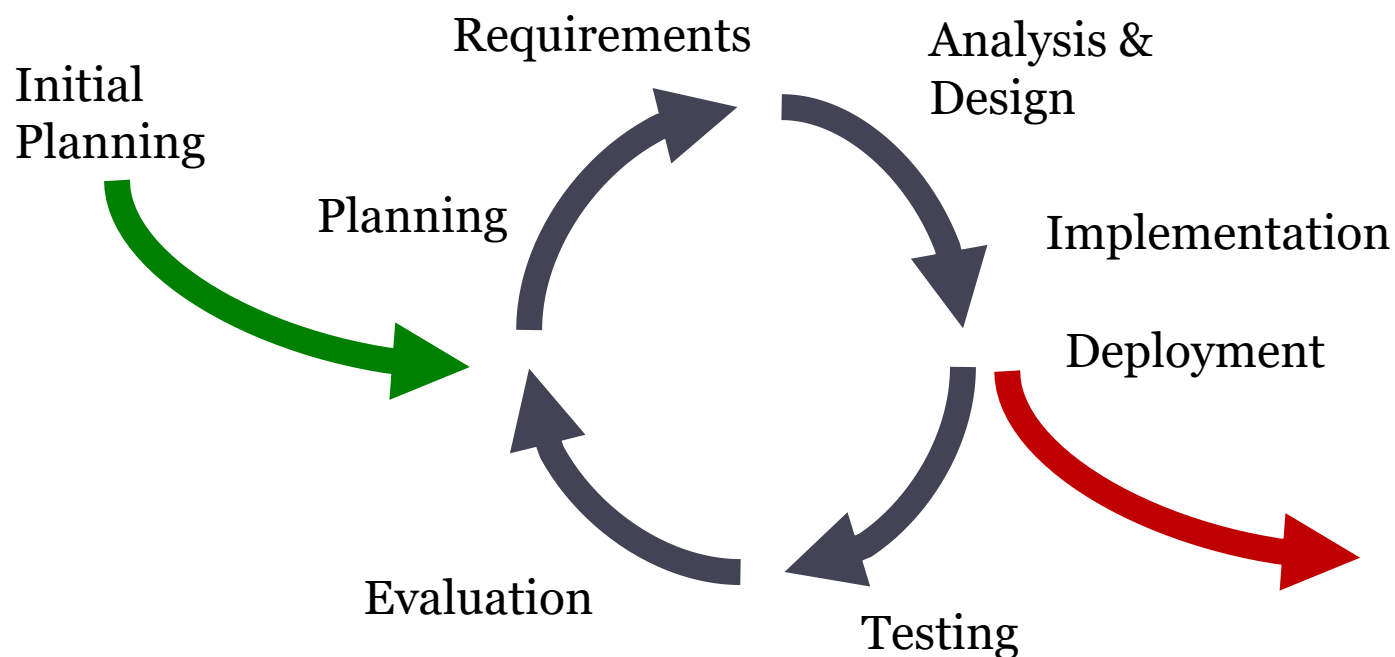
Software implemented but unused



# Iterative Models

Based on Prototypes

Risk assessment after each iteration



# Agile methodologies Overview

# Agile methodologies

## Lots of variants

RAD ([www.dsdm.org](http://www.dsdm.org), 95)

SCRUM (Sutherland & Schwaber, 95)

XP - eXtreme Programming (Beck, 99)

Feature driven development (DeLuca, 99)

Adaptive software development (Highsmith, 00)

Lean Development (Poppendieck, 03)

Crystal Clear (Cockburn, 04)

Agile Unified Process (Ambler, 05)

. . .



# Agile methods

Agile Manifesto ([www.agilemanifesto.org](http://www.agilemanifesto.org))

Individuals and  
interactions

over

Processes and  
Tools

Working  
Software

over

Comprehensive  
Documentation

Customer  
Collaboration

over

Contract  
Negotiation

Responding  
to change

over

Following a  
Plan

# Agile methods

## Feedback

Changes of code are OK during development

## Minimize risk

Software in short intervals

Iterations of days

Each iteration takes all the development cycle

# Some agile principles (XP)

1. Adapt to change
2. Testing
3. Pair programming
4. Refactoring
5. Simple design
6. Collective code ownership
7. Continuous integration
8. On-site customer
9. Small releases
10. Sustainable pace
11. Coding standards

# Adopt change

After each iteration, update plans  
Requirements through user stories

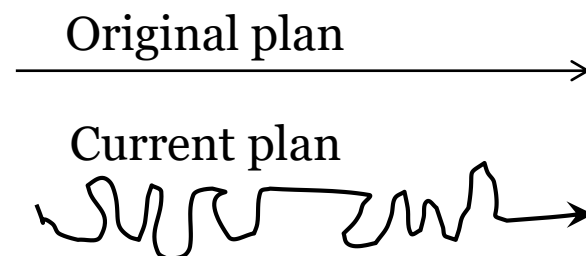
Short descriptions (size of a card)

Goals ordered by usnig according to priority

Risk and resources estimated by developers

User stories = acceptance testing

Wellcome changing requirements



# TDD - Test driven development

Write a test before coding

Initially, code will fail

Goal: pass the test

Result:

Automated set of tests

Easier refactoring



# Different types of testing

## Unit testing

Check each unit separately

## Integration testing

Smoke testing

## Acceptance testing

Check with user stories

## Performance/capacity testing:

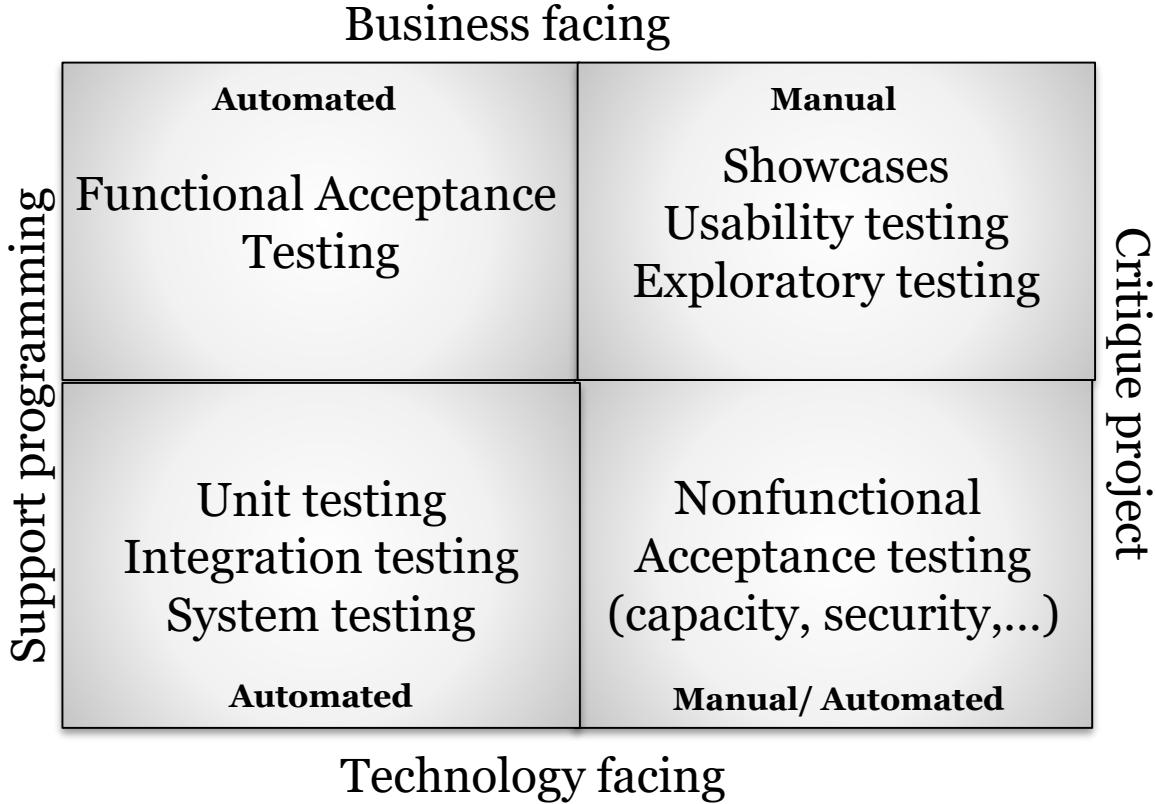
Load testing

## Regression testing

Check that new changes don't introduce new bugs, or  
*regressions*



# Types of testing



# Acceptance testing

## Behavior-driven development (BDD)

Tests come from user stories

They can be written collaboratively with the client

Tools: Cucumber, JBehave, Specs2,...

Tests act as contracts

Can also be used to measure progress

**Feature:** Buscar cursos

Para mejorar el uso de los cursos

Los estudiantes deberían ser capaces de buscar cursos

**Scenario:** Búsqueda por asunto

**Given** hay 240 cursos que no tienen el asunto "Biología"

**And** hay 2 cursos A001, B205 que tienen el asunto "Biología"

**When** Yo busco el asunto "Biología"

**Then** Yo debería ver los cursos:

| Código |

| A001 |

| B205 |

# Testing: FIRST Principles

F - Fast

Execution of (subsets of) tests must be quick

I - Independent:

No tests depend on others

R - Repeatable:

If tests are run N times, the result is the same

S - Self-checking

Test can automatically detect if passed

T - Timely

Tests are written at the same time (or before) code

# Test doubles

*Dummy* objects:

Objects that are passed but not used

*Fake* objects: Contain a partial implementation.

*Stubs*: contain specific answers to some requests

*Spies*: *stubs* that record information for debugging

*Mocks*: mimic the behavior of the real object

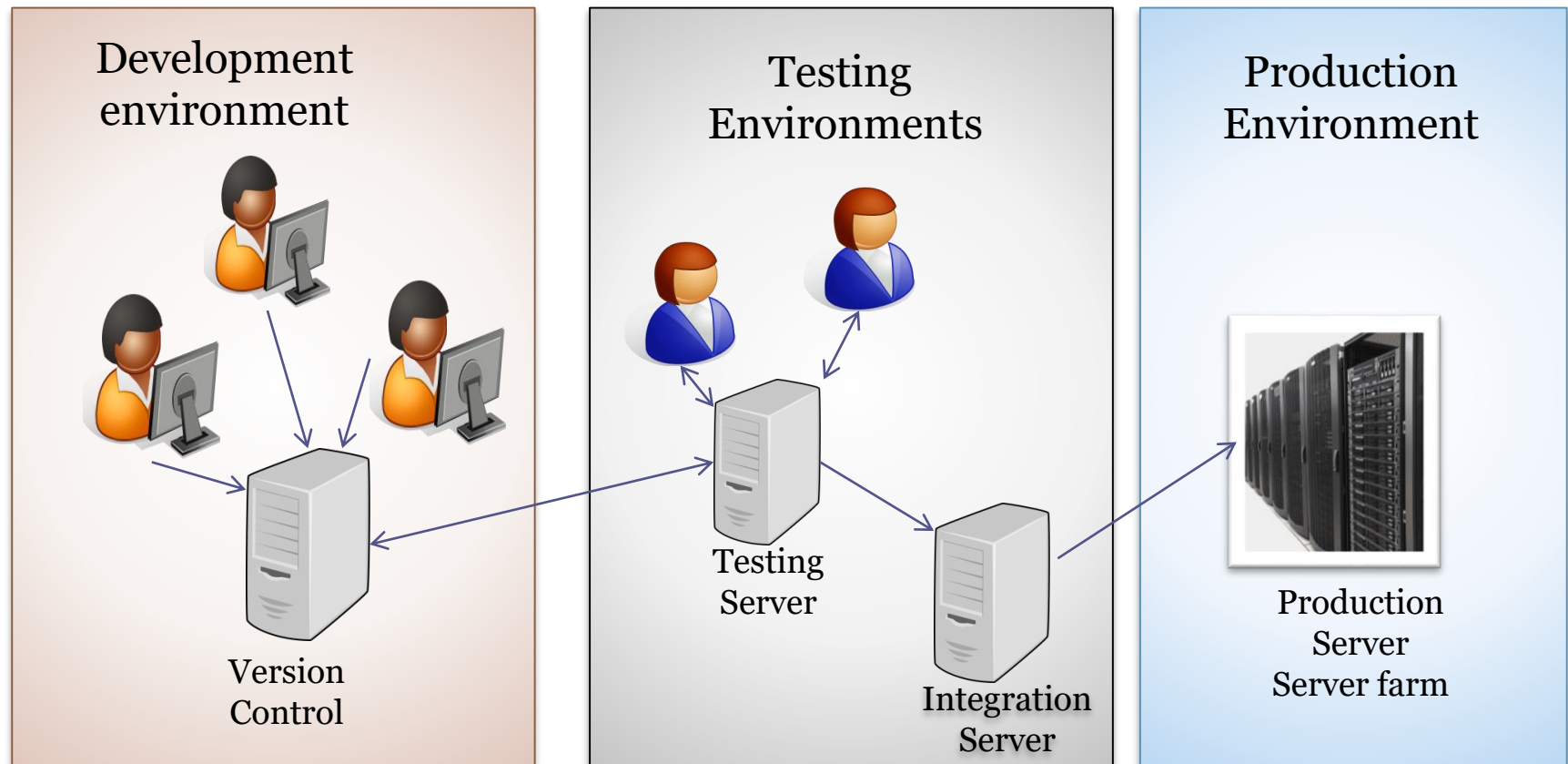
Mocks may contain assertions about the order/number of times methods are called

**Fixtures**: Tools that support tests

Testing databases, some files, etc.



# Environments



A staging environment is also used

# Pair programming & Code reviews

2 software engineers work together

*Driver* manages keyboard and creates implementation

*Observer* identifies failures and gives ideas

Roles are exchanged after some time

Pull requests: Before accepting changes, code can be reviewed





# Simplicity

Favor Simple design

Reaction to Big Design Up Front

Obtain the simpler design that works

Automated documentation

JavaDoc and similar tools



# Refactoring

Improve design without changing functionality

Simplify code (eliminate redundant code)

Search new opportunities for abstraction

Regression testing

Based on the test-suite



# Collective ownership of code

Code belongs to the project, not to some engineer  
Engineers must be able to browse and modify any part of the code

Even if they didn't wrote it

Avoid code fragments that only one person can modify



# Continuous Integration

Frequently integrating one's new or changed code with the existing code repository

Running all unit and integration tests

Merge all developer working copies

## Goals

Help Test Driven Development

Maintain all programmers code up to date

Avoid integration hell



# Continuous Integration

## Best practices:

- Maintain code repository

- Automate the build

- Make the build self testing

- Everyone commits to the baseline

- Every commit should be built

- Keep the build fast

- Test in a clone of the production environment

- Make it easy to get the latest deliverables

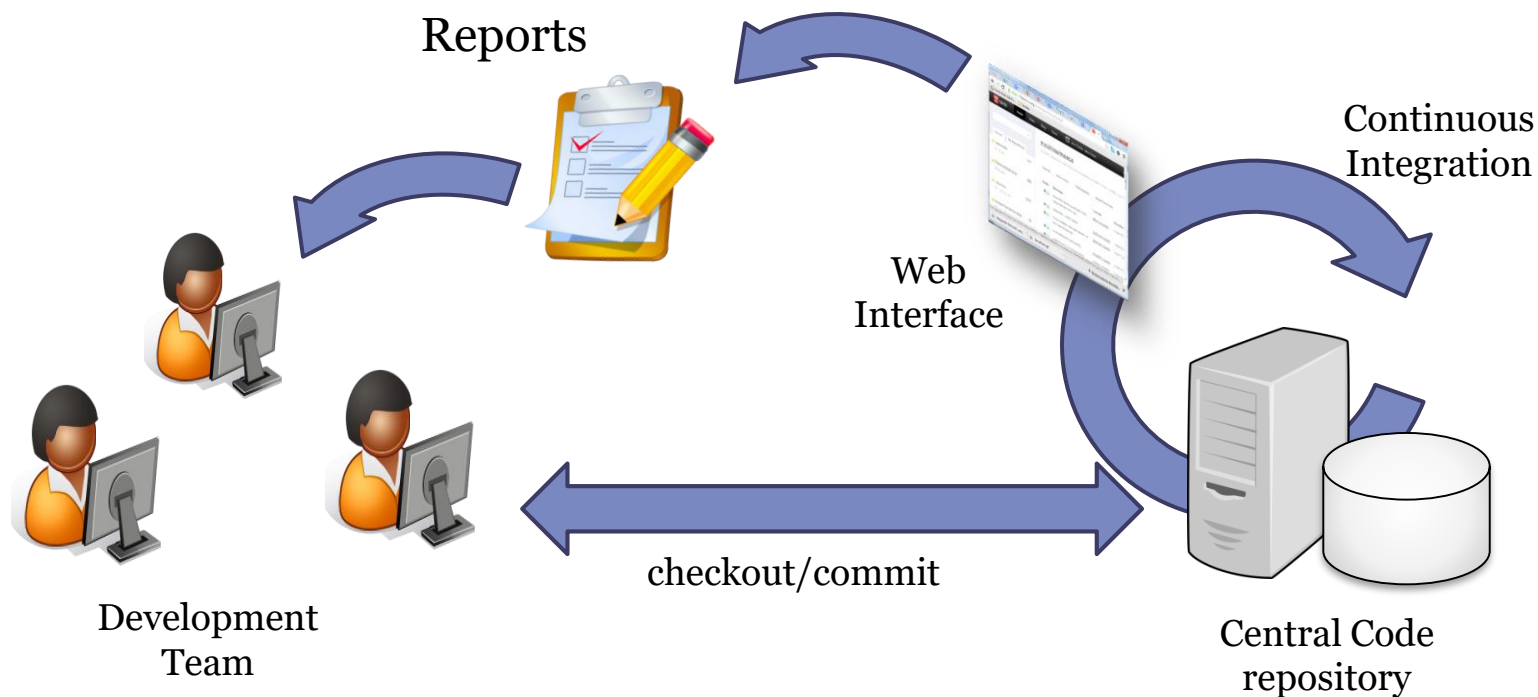
- Everyone can see the results of the latest build

- Automate deployment

# Continuous integration

## Continuous integration tools

Hudson, Jenkins, Travis, Bamboo





# On-place customer

Customer available to clarify user stories and help taking critical business decisions

## Advantages

Developers don't do guesses

Developers don't have to wait for decisions

Improves communication



# Continuous delivery

## Small releases

Small enough while offering value to the user

Releases are not, for example, implement DB

Obtain feedback soon from client

Plan next release after each iteration

Try to release something every week



# Sustainable pace

Avoid extra-work loads

40h/week = 40h/week

Tired programmers write bad code

It will slow the development at long time



# Clean code & code conventions

Facilitate code refactoring by other people

Use good practices

Code styles and guidelines

*Avoid code smells*

*software craftsmanship manifest*

*Clean Code (Robert C. Martin)*



# Some agile methods

## Variants

### Scrum

Project/people management

Divide work in sprints

15' daily meetings

Product Backlog

### Kanban

*Lean model*

Just in Time Development

Limit workloads



# Configuration Management

## Different software versions

- New or different functionalities

- Issues and bugs management

- New execution environments

## Configuration management

- Manage software evolution

- System changes = team activities

- Imply cost and effort

# Version control

Systems that manage different code versions

- Be able to Access all the system versions

- Easy to rollback

- Differences between versions

- Collaborative development

- Branch management

- Metadata

- Author of a version, update date, who to blame, etc.

# Releases and versions

Version: instance of a system which has a different functionality to other instances

Release (deliverable): instance of a system which is distributed to external people outside to development team.

It can be seen as a final product at some point





# Version naming - some conventions

Pre-alpha

Before testing

Alpha

During testing

Beta (or prototype)

Testing made by some users

Beta-tester: user that does the testing

Release-candidate

Beta version that could become final product

# Other schema namings

## Using some attributes

Date, creator, language, client, state,...

## Recognizable Names

Ganimede, Galileo, Helios, Indigo, Juno,...

Precise Pangolin, Quantal Quetzal,...

## Semantic Versioning (<http://semver.org>)

MAJOR.MINOR.PATCH (2.3.5)

MAJOR: changes incompatible with previous versions

MINOR: new functionality compatible with previous versions

PATH: Bugfix compatible with previous versions

Version 0 (inestable)

Pre-releases (names added at the end): 2.3.5-alpha

# Publishing releases

A *release* implies functionality changes

## Planning

Publishing a release has costs

Usually, current users don't want new releases

External factors:

Marketing, clients, hardware, ...

Agile model: frequent *releases*

Continuous integration minimizes risk

# Publishing Releases

A release is more than just software

Configuration files

Some needed data files

Installation programs

Documentation

Publicity and packaging

**Distribution channels:**

Physical (CDs, DVDs), Web (download), stores

...See later...

# Continuous delivery

## Continuous delivery

Frequent releases to obtain feedback as soon as possible

TDD & continuous integration

Deployment pipeline

Advantages:

Embrace change

Minimize integration risks



### **Wabi-sabi philosophy**

Accept imperfection

Software that is not finished: Good enough

# DevOps

Merge ***development and operations***

Cultural change where the same team participates in:

Code: Development and code review, continuous integration

Build: Version control, building and integration

Test

Package: Artifact management

Release: version automation

Configuration and management

Monitorization: performance, user experience

# Construction tools

# Construction languages

## Configuration languages

Resource definitions (Json, XML, Turtle)

Examples: .travis.yml, package.json, pom.xml

## Scripting languages

Shell/batch scripts

## Lenguajes de programación

Examples: Java, Javascript,...

## Visual languages

Examples: scratch, blender, ...

## Formal

Examples: B-trees, Z language, OCL, ...



# Coding aspects

## Naming conventions

Important for other programmers, maintainers...

Classes, types, variables, named constants, ...

Error handling

Source code organization

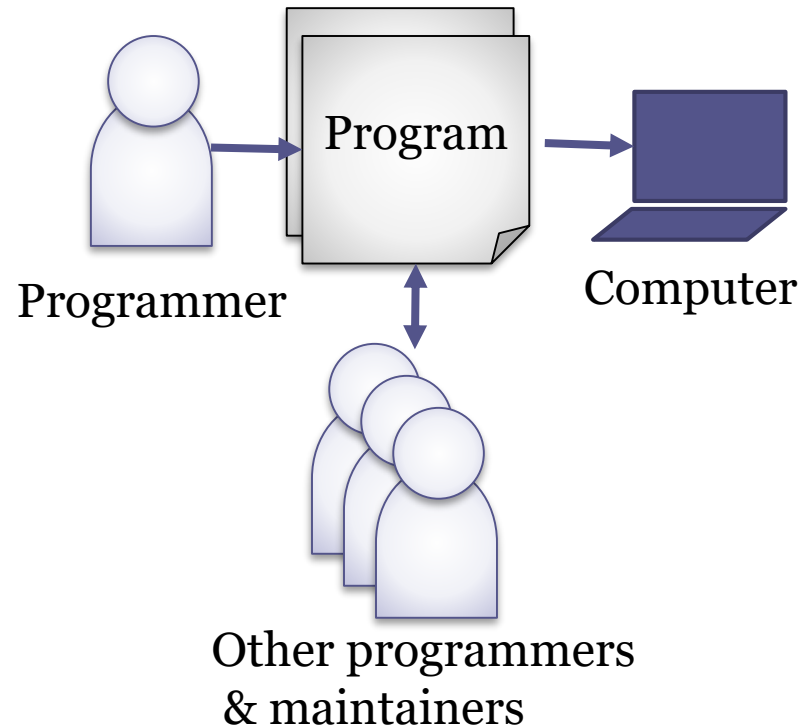
Packages, folders, ...

Dependencies

Libraries imported

Code documentation

Javadocs, jsdoc...



# Testing

Unit testing

Integration testing

Load testing

Regression testing

. . .

Best practice:

Separate testing code and dependencies from  
production code

# Construction for reuse

## Parameterization

Add parameters

Common error: magical numbers in code

Configuration/resource files

## Conditional compilation

## Encapsulation

Separate interface from implementation

Common error: internal parts public in libraries

## Packaging

Common error: manual tasks for packaging

## Documentation

API documentation

# Construction with reuse

## Selection of reusable units

Externally developed components (COTS, FOSS)

## Handling dependencies

<See later>

## Handling updates

What happens when other libraries are updated?

## Legal issues

Can I really use that library?

For commercial software?

Be careful with GNU libraries

Is the library well maintained?

# Construction tools

## Text editors

vi, emacs, Visual Studio Code, Sublime,....

## Integrated Development Environments (IDEs)

Examples: IntelliJ, Eclipse

## Graphical User Interface (GUI) builders

Android Studio UI Editor, QtEditor,...

## Quality assurance (QA) tools

Test, analysis, ...<See next slide>

# Software Quality Assurance

## Tests

xUnit, test frameworks (mocha)

Assertion languages (chai)

Test coverage tools

## Assertions

Pre-conditions asserted on methods

## Inspections & code reviews

Pull requests with code reviews

## Code Analysis tools

<See next slide>

# Code analysis tools

## Static vs dynamic code analysis

Without running the code (or at runtime)

Examples: PMD, SonarCube,... (Codacy)

## Debuggers

Interactive vs static, Tracers & logging

## Profilers

Information about resource usage

Memory, CPU, method calls, etc.

## Test coverage tools

Report which lines of code have been run during tests

## Program slicing

Program fragment (slice) that has been run

Examples: CodeSurfer, Indus-kaveri,...

# Control version systems



# Version control

Systems that manage different code versions

- Be able to Access all the system versions

- Easy to rollback

- Differences between versions

- Collaborative development

- Branch management

- Metadata

- Author of a version, update date, who to blame, etc.

# Releases and versions

Version: instance of a system which has a different functionality to other instances

*Release* (deliverable): instance of a system which is distributed to external people outside to development team.

It could be considered a final product at some point of time



# Version naming - some conventions

Pre-alpha

Before testing

Alpha

During testing

Beta (or prototype)

Testing made by some users

Beta-tester: user that does the testing

Release-candidate

Beta version that could become final product

# Other schema namings

## Simple numbering

1.1, 1.1-alpha

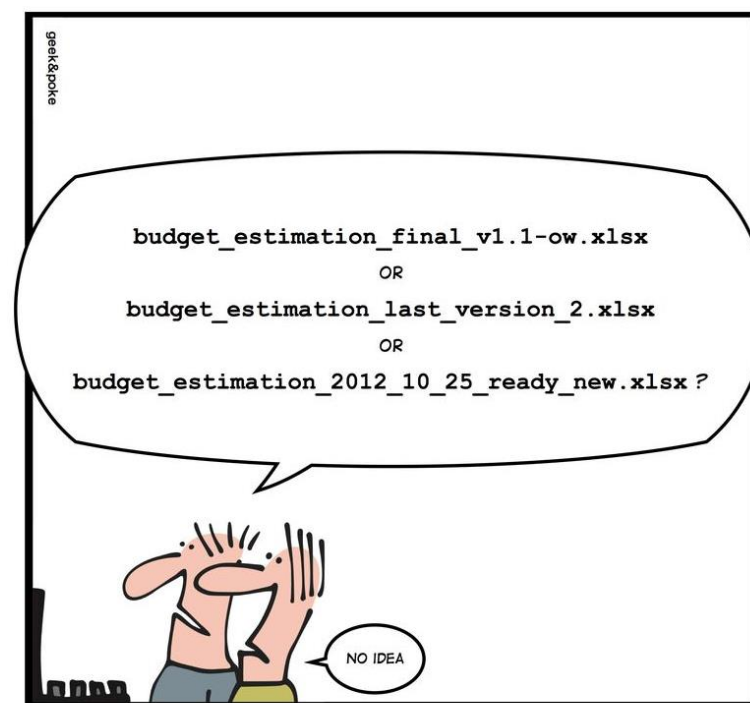
## Using attributes

Date, creator, language, client, state,...

## Recognizable Names

Ganimede, Galileo, Helios, Indigo, Juno,...

Precise Pangolin, Quantal Quetzal,...



# Publishing releases

A *release* implies functionality changes

## Planning

Publishing a release has costs

Usually, current users don't want new releases

External factors:

Marketing, clients, hardware, ...

Agile model: frequent *releases*

Continuous integration minimizes risk

# Publishing Releases

A release is more than just software

Configuration files

Some needed data files

Installation programs

Documentation

Publicity and packaging

**Distribution channels:**

Physical (CDs, DVDs), Web (download), stores

# Version control

## Definitions

**Repository:** where changes are stored

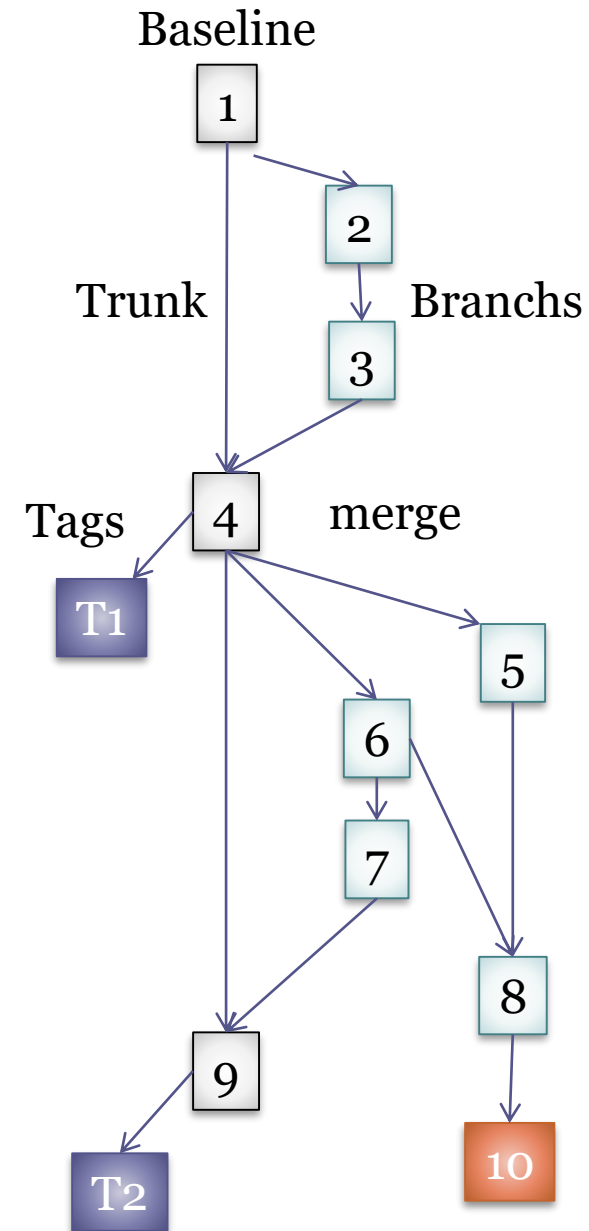
**Baseline:** Initial version

**Delta:** changes from one version to other

**Trunk (master):** Main branch in a system

**Branch:** deviation from main branch

**Tag:** Marks a line of versions



# Version control

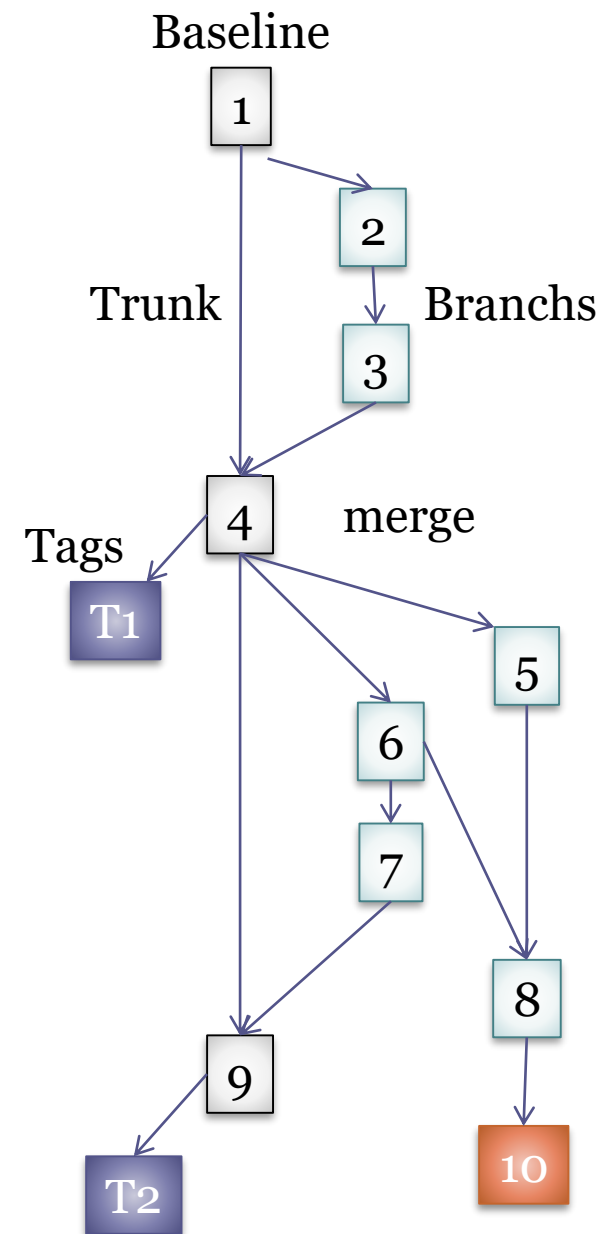
## Definitions

**Checkout:** Working Local copy from a given branch

**Commit:** Introduce current changes in the control version system.

*Merge:* Combine two sets of changes

**Branching styles:** by feature, by team, by version





# Version control

## 2 types

### Centralized

- Centralized repository for all the code

- Centralized administration

- CVS, Subversion, ...

### Distributed

- Each user has its own repository

- Git, Mercurial

# Git

Designed by Linus Torvalds (Linux), 2005

Goals:

- Applications with large number of source code files

  - Efficiency

- Distributed work

  - Each development has its own repository

    - Local copy of all the changes history

    - It is possible to do commits even without internet connection

- Support for non-linear development (branching)



# Local components

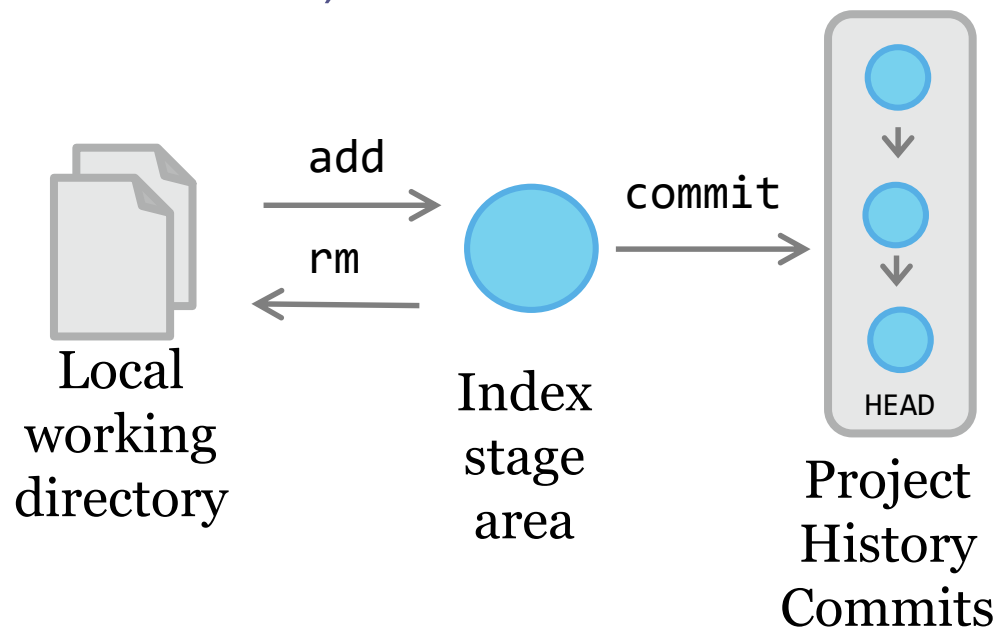
## 3 local components:

Local working directory

Index (stage area). Also called cache

Project history: Stores versions or commits

HEAD (most recent version)



# Branches

Branches

feature-2   feature-1   develop   hotfix-1   master   tags

Git facilitates branch management

master = initial branch

Operations:

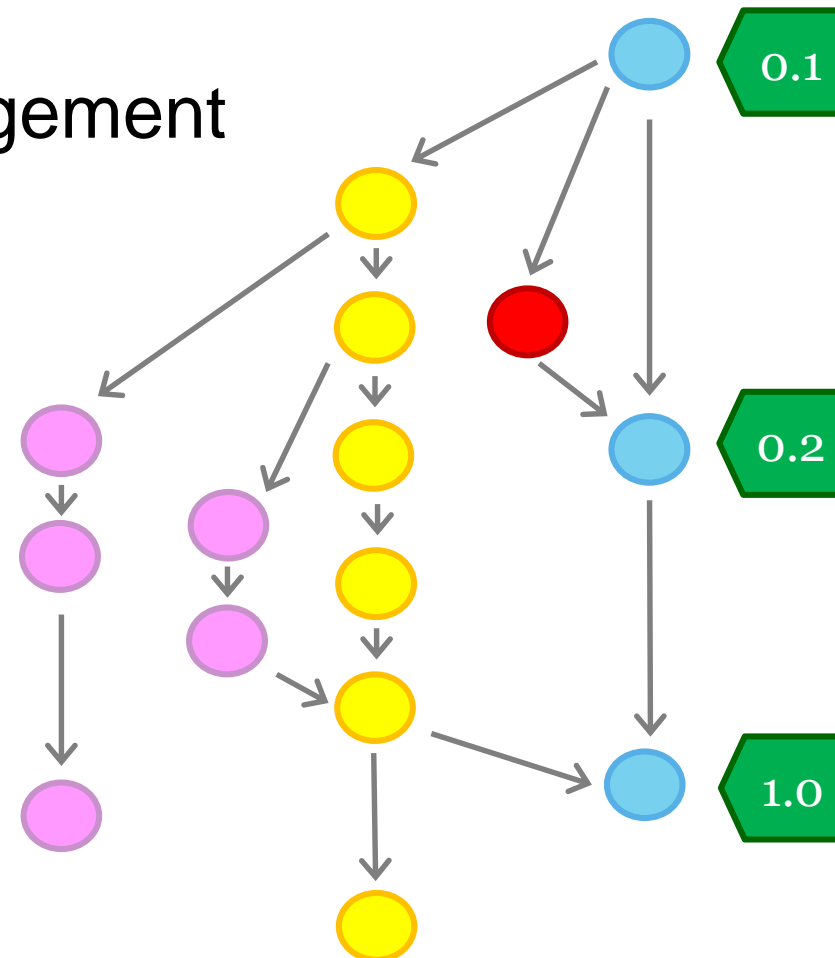
Create branches (*branch*)

Change branch (*checkout*)

Combine (*merge*)

Tag branches (*tag*)

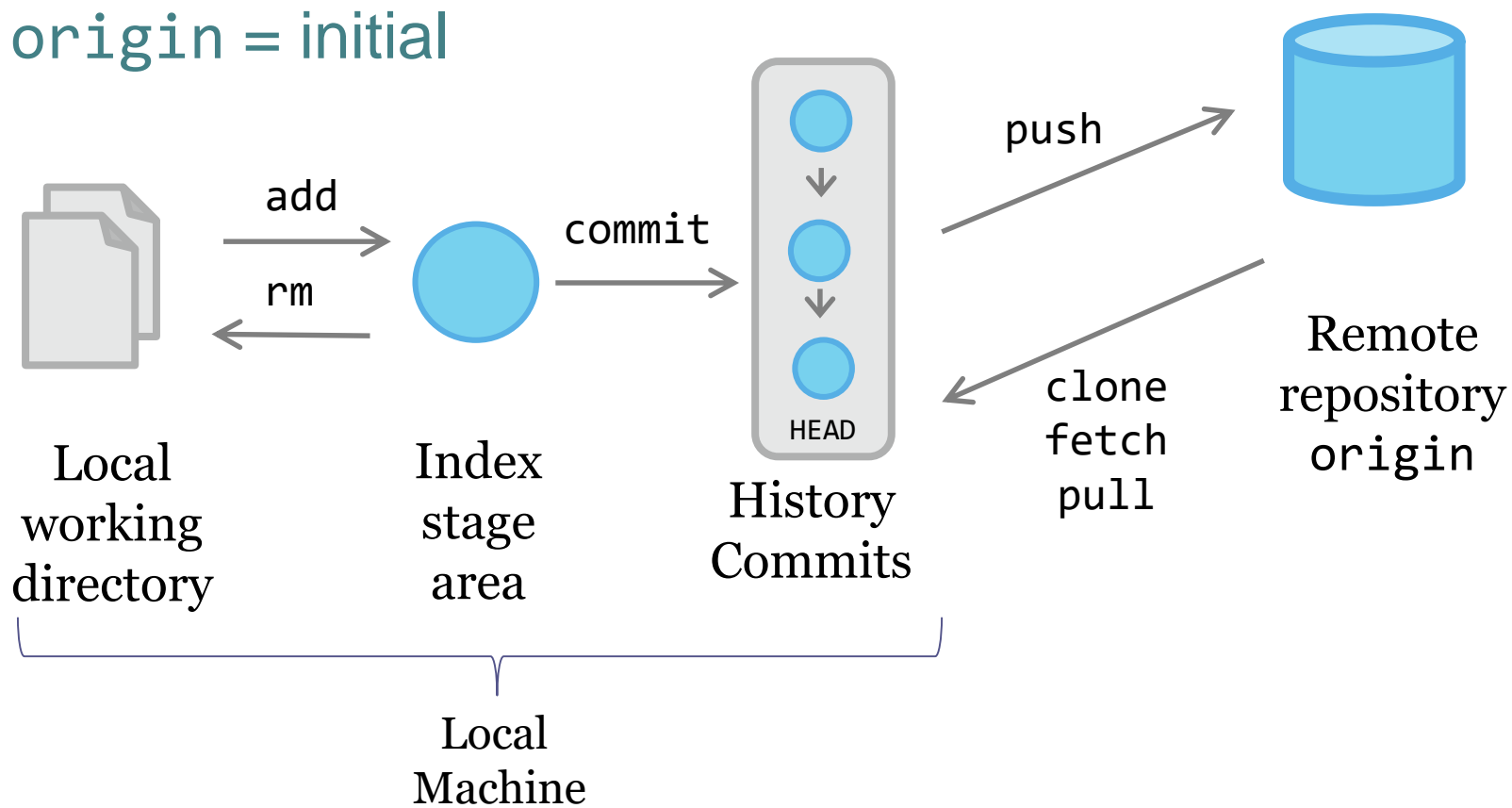
Multiple branching styles



# Remote repositories

Connect with remote repositories

origin = initial



# Git - usual commands

**init**: initialize repository

**clone**: clone a repository

**add**: add contents to index

**commit**: record changes

**status**: check state of repository

**pull**: get changes from an external repository

**push**: put changes in an external repository

**branch**: manage branches (list, create, delete)

**merge**: combine two branches

**checkout**: recover a given branch

**rm**: delete files

**mv**: move files

**.gitignore** file indicates which files are not going to be tracked by version control system

# Dependency management

# Dependency management

Library: Collection of functionalities used by the system that is being developed

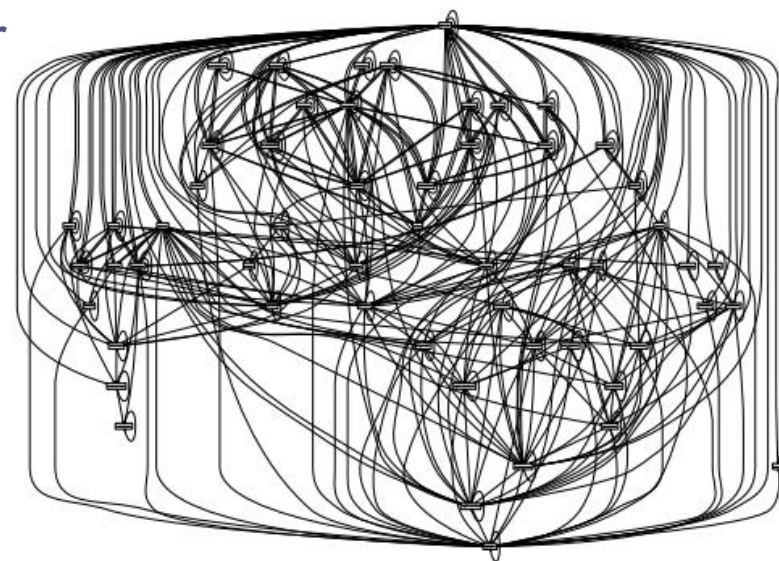
System depends on that library

Library can depend on other libraries

Library can evolve

Incompatible versions appear

Dependency graph



Mozilla Firefox dependency graph

Source: The purely functional deployment model. E. Dolstra (PhdThesis, 2006)



# Dependency graph

Graph  $G = (V, E)$  where

$V$  = Vertex (components/packages)

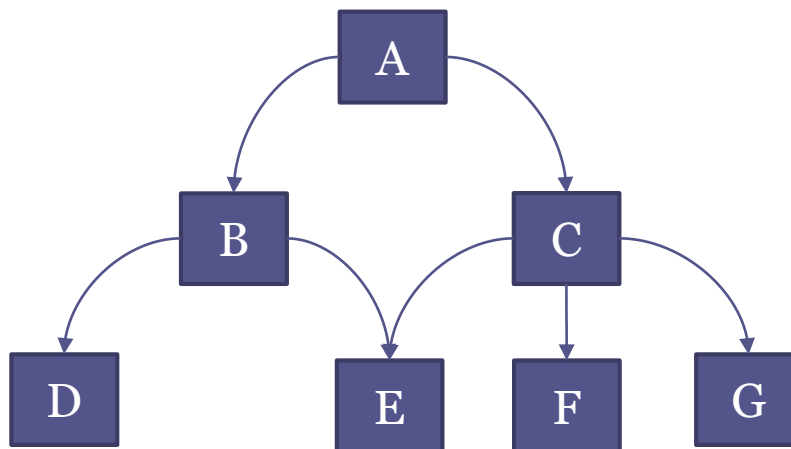
$E$  = Edges  $(u, v)$  that indicate that  $u$  depends on  $v$

**CCD metric** (cumulative component dependency)

Sum of every component dependency

Each component depends on itself

In the example:  
 $CCD = 7 + 3 + 4 + 1 + 1 + 1 + 1 = 18$



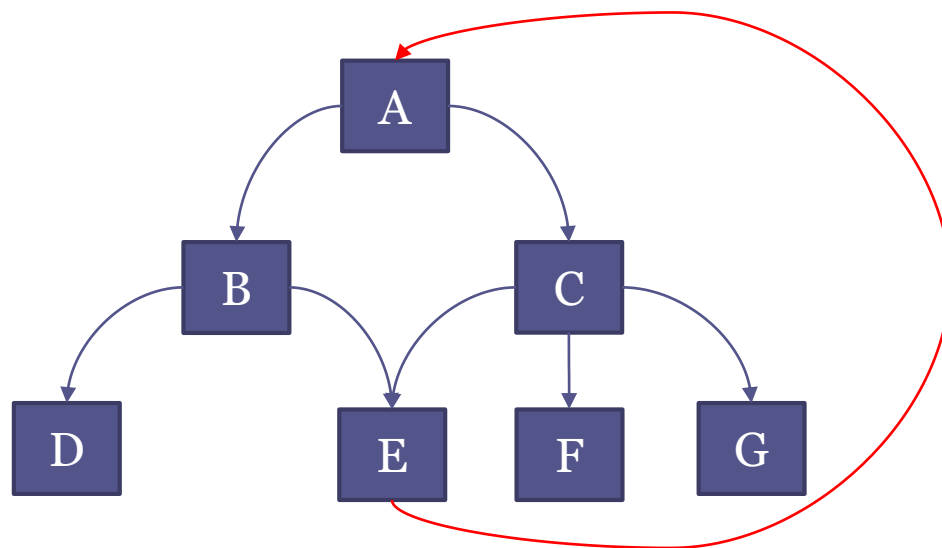
# Cyclic dependencies problem

The dependency graph should not have cycles

Adding a cycle can damage CCD

Example:

$$\text{CCD} = 7+7+7+1+7+1+1=31$$



# Dependency management

## Different models

Local installation: libraries are installed for all the system

Example: Ruby Gems

Embed external libraries in the system (version control)

Ensures a correct version

External link

External repository that contains the libraries

Depends on Internet and on library evolution

# Build management

Automate building tasks

Some quality attributes:

Correctness:

- Avoid mistakes (minimize "*bad builds*")

- Eliminate repetitive and redundant tasks

Simplicity: Handle complexity

Automation & releasability

- Have history of builds and releases

- Continuous integration

Cost

- Save time & money

# Types of build automation

## On-demand

A user running a script at the command line

## Scheduled

Automatically run at certain hours

Continuous integration server

Example: nightly builds

## Triggered

At every commit to a version control system

Continuous integration server linked to version control system

# Build Automation Tools

Makefile (C world)

Ant (Java)

Maven (Java)

SBT (Scala, JVM languages)

Gradle (Groovy, JVM languages)

rake (Ruby)

npm, grunt, gulp (Javascript)

etc.

# Automate building

**make:** Included in Unix

Product oriented

Declarative language based on rules

When the Project is complex, configuration files can be difficult to manage/debug

Several versions: BSD, GNU, Microsoft

Very popular in C, C++, etc.

# Automate building

**ant**: Java Platform

Task oriented

XML syntax (build.xml)



# Automate building

## maven: Java Platform

Convention over configuration

Manage project lifecycle

Dependency management

XML syntax (pom.xml)

# Automate building

## Embedded languages

Domain specific languages embedded in higher level ones

Great versatility

Examples:

`gradle` (Groovy)

`sbt` (Scala)

`rake` (Ruby)

`Buildr` (Ruby)

`gulp` (Javascript)

...

# New tools

Pants (Foursquare, twitter)

<https://pantsbuild.github.io/>

Bazel (Google)

<http://bazel.io/>

Buck (Facebook)

<https://buckbuild.com/>

# Maven

Build automation tool

Describes how software is built

Describes software dependencies

Principle: Convention over configuration



Jason van Zyl  
Creator of Maven

# Maven

Typical development phases:

clean, compile, build, test, package, install, deploy

Module identification

3 coordinates: Group, Artifact, Version

Dependencies between modules

Configuration: XML file (Project Object Model)

pom.xml

# Maven

## Artifact repositories

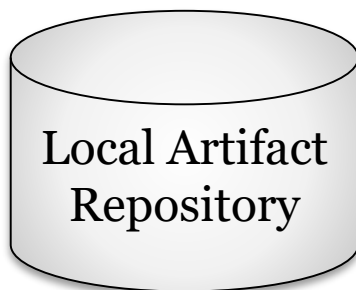
Store different types of artifacts

JAR, EAR, WAR, ZIP, plugins, etc.

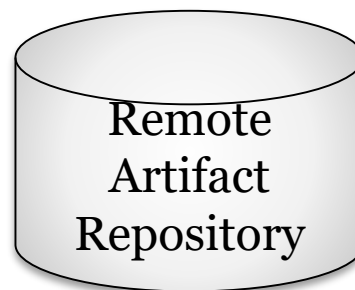
Every interaction is made through the repository

No relative paths

Share modules between development teams



`<user>/ .m2/repository`



Maven Central

# Maven Central

Public repository of projects

Over 1 mill GAV

≈ 3000 new projects each month (GA)

≈ 30000 new versions each month(GAV)\*

 The Central Repository

<http://search.maven.org/>

Other repositories:

<https://bintray.com/>

\* Source: <http://takari.github.io/javaone2015/still-rocking-it-maven.html>

# POM - Project Object Model

XML syntax

Describes a project

- Name and version

- Artifact type (jar, pom, ...)

- Source code localizations

- Dependencies

- Plugins

- Profiles

  - Alternative build configurations

Inheritance structure

Reference: <https://maven.apache.org/pom.html>



# POM - Project Object Model

## Inheritance structure

### Super POM

- Maven's default POM

- All POMs extend the Super POM unless explicitly said

### parent

- Declares the parent POM

- Dependencies and properties are combined

# Maven

## Project identification

### GAV (Group, Artifact, Version)

Group: grouping identifier

Artifact: Project name

Version: Format {Major}.{Minor}.{Maintenance}

It is possible to add "-SNAPSHOT" (in development)

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>es.uniovi.asw</groupId>
  <artifactId>censusesN</artifactId>
  <version>0.0.1</version>
  <name>censusesN</name>
  ...
</project>
```

# Maven

## Folder structure

Maven uses a conventional structure

src/main

src/main/java

src/main/webapp

src/main/resources

src/test/

src/test/java

src/test/resources

...

Output directory:

target

# Maven Build life cycle

## 3 built-in lifecycles

default

Project deployment

clean

Project cleaning

site

Project's site documentation

Each life cycle has some specific phases

# clean

Clean compiled code and other stuff

3 phases

pre-clean

clean

post-clean

# default lifecycle

Compilation, testing and deploying  
Some phases

```
validate  
initialize  
generate-sources  
generate-resources  
compile  
test-compile  
test  
package  
integration-test  
verify  
install  
deploy
```

# site lifecycle

Generates Project's site documentation  
Phases

```
pre-site  
site  
post-site  
site-deploy
```

# Maven

Automatic dependency management

GAV identification

Scopes

compile

test

provide

Type

jar, pom, war,...

```
...  
<dependency>  
<groupId>commons-cli</groupId>  
<artifactId>commons-cli</artifactId>  
<version>1.3</version>  
</dependency>  
...
```



# Maven

Automatic dependency management

Dependencies are downloaded

Stored in a local repository

We can create intermediate repositories (proxies)

Examples: common artifacts for some company

Transitivity

A depends on B

B depends on C

⇒ If a system depends on A

Both B and C are downloaded

# Maven modules: aggregation

Big projects can be decomposed in subprojects

Each Project creates one artifact

Contains its own `pom.xml`

Parent Project groups modules

```
<project>
  ...
  <packaging>pom</packaging>
  <modules>
    <module>extract</module>
    <module>game</module>
  </modules>
</project>
```

# Maven Plugins

Maven architecture based on plugins

2 types of plugins

build

reporting

List of plugins: <https://maven.apache.org/plugins/index.html>

# Maven

## Other phases and plugins

`archetype:generate` - Generates Project archetype

`eclipse:eclipse` - Generates eclipse project

`site` - Generates Project web site

`site:run` - Generates Project web site and starts server

`javadoc:javadoc` - Generates documentation

`cobertura:cobertura` - Reports code executed during tests

`checkstyle:checkstyle` - Check coding style

`spring-boot:run` - Run a spring application

npm

# npm

Node.js package manager

Initially create by Isaac Schlueter

Later became Npm inc.

Default package manager for NodeJs

Manages dependencies

Allows scripts for common tasks

Software registry

Public or paid packages

Configuration file: package.json

# npm configuration: package.json

Configuration file: package.json

npm init creates a simple skeleton

Fields:

```
{
  "name":           "...mandatory...",
  "version":        "...mandatory...",
  "description":    "...optional...",
  "keywords":       "...",
  "repository":     { ... },
  "author":         "...",
  "license":        "...",
  "bugs":           { ... },
  "homepage":       "http://. . .",
  "main":           "index.js",
  "devDependencies": { ... },
  "dependencies":   { ... },
  "scripts":        { "test": " ... " },
  "bin":            { ... },
}
```

Note: Yeoman provides fully featured scaffolding

# npm packages

Repository: <http://npmjs.org>

Installing packages:

2 options:

Local

```
npm install <packageName> --save (--save-dev)
```

Global

```
npm install -g <packageName>
```



# npm dependencies

## Dependency management

Local packages are cached at `node_modules` folder

Access to modules through: `require('...')`

Global packages (installed with `--global` option)

Cached at: `~/.npm` folder

Scoped packages marked by `@`

# npm commands and scripts

Npm contains lots of commands

start  $\approx$  node server.js

test  $\approx$  node server.js

ls lists installed packages

...

Custom scripts:

run-script <name>

<https://docs.npmjs.com/cli-documentation/>