Software Development Processes

Development phases, waterfall, agile methodologies, terminology, best practices

Software lifecycles

Software lifecycles can be described by phases:

- **Analysis of requirements**. What is the software supposed to do?
- **Design**. How is it supposed to do it?
- **Implementation**. Coding individual components and integrating them together.
- **Testing**. Attempting to detect errors.
- **Integration**. Pulling all the software components into a whole.
- **Deployment**. Release to customers.
- **Maintenance**. Ongoing bug fixing and enhancement.

Software artifacts

- **Software artifact**: any document, file, or tangible creation related to the software or its creation or maintenance. Includes source code, executables, documentation (requirements, design, software, end-user).
- **Business analysts** create analysis artifacts: requirements documents, **use cases** (step-by-step description of how a system provides a service for a user), **user stories** (short description of required function from the user's point of view).
- **Software architects** create design artifacts, which include UML diagrams.
- Software developer artifacts: code, source documentation, config files, executables, bug documentation, APIs.

Rq Requirements/Analysis

Analysis answers the question “*What* is the program supposed to do?”

- During analysis, the requirements are fleshed out.
- Typically done with multiple customer interviews.
- For real-world software projects, analysis can be difficult, tedious, and unlikely to "get it right," especially the first time.

Software Artifacts for analysis:

- Input: Description of the overall problem. Business requirements documents (why is this software needed?).
- Output: use cases, use case diagrams, and user stories.

A **user story** is a description of a desired feature taken from the viewpoint of a user. It tells what the user wants and why.

In commercial or large scale environments, analysis artifacts such as user stories are maintained in a project repository designed to track them. Example: JIRA.

Design

Design answers the question “*How* is the program supposed to do it?”

- This is where the essential model for the solution is created.
- During design, the software components are selected (including databases, web servers, etc).
- Major modeling tools: most of the UML diagrams.

Software Artifacts for design:

- Input: Use cases, user stories.
- Output: All other UML diagrams, written descriptions, resource specifications (people/equipment/platforms & languages/supporting systems such as databases).

Implementation

Implementation translates the design into code.

- Some language choices are easier to work with than others.
- Coding is usually done to shop standards.
- Coding is almost never done by a single person.
- Part of coding is integrating the major parts into a coherent system.
- Collaboration tools include version control software (Git, CVS, etc)

Software artifacts for implementation:

- Input: All artifacts produced for design (UML diagrams, etc).
- Output: Source code, documentation, configuration files, media, executables/deployment packages, source code repository commits and comments, feature/bug tracking notations.

Testing

The goal of testing is to examine and execute software with the intent of finding errors.

- The testing process can be done from the outset. You can start planning testing when you are analyzing the requirements.
- At a minimum, testing involves writing test cases, running them, and documenting them.

Software artifacts for testing:

- Input: Notification (email/system) of project startup, use cases and user stories, requirements documents.
- Output: Test code (including test harnesses, scaffolding, etc), bug and test database entries, test cases: input, output, documentation of expected and actual results.

Dp

Deployment

The goal of deployment is to deliver software to the customer.

Software engineers package the software and associated documentation. Other support includes such things as training.

- Official software release. Examples:
  - Commercial/closed source: MS Word
  - Open Source: Eclipse
  - RedHat release for installation on large infrastructures.
- Deployment for CSC 216 means submitting your programming assignment for grading.

Software artifacts for deployment

- Input: Completed software, test results
- Output: Executables, installs, documentation, APIs, etc.

Maintenance

The goal of maintenance is to upgrade and fix the software after it has been delivered. Includes:

- Bug fixing (post release). Maintenance requires understanding the code base.
- Enhancement.
- Changing requirements (can be anything, such as functionality, supported platforms, database changes).
- Updating to more/newer operating systems, web browsers, databases, etc.

Software artifacts for maintenance:

- Input: Everything, plus bug reports and enhancement requests.
- Output: Same as for deployment plus notations in bug/enhancement tracking systems.

Waterfall vs agile methodologies

Waterfall program development is a traditional approach to program development. With waterfall, one phase is completed before the next begins. Go back to the previous phase if necessary.

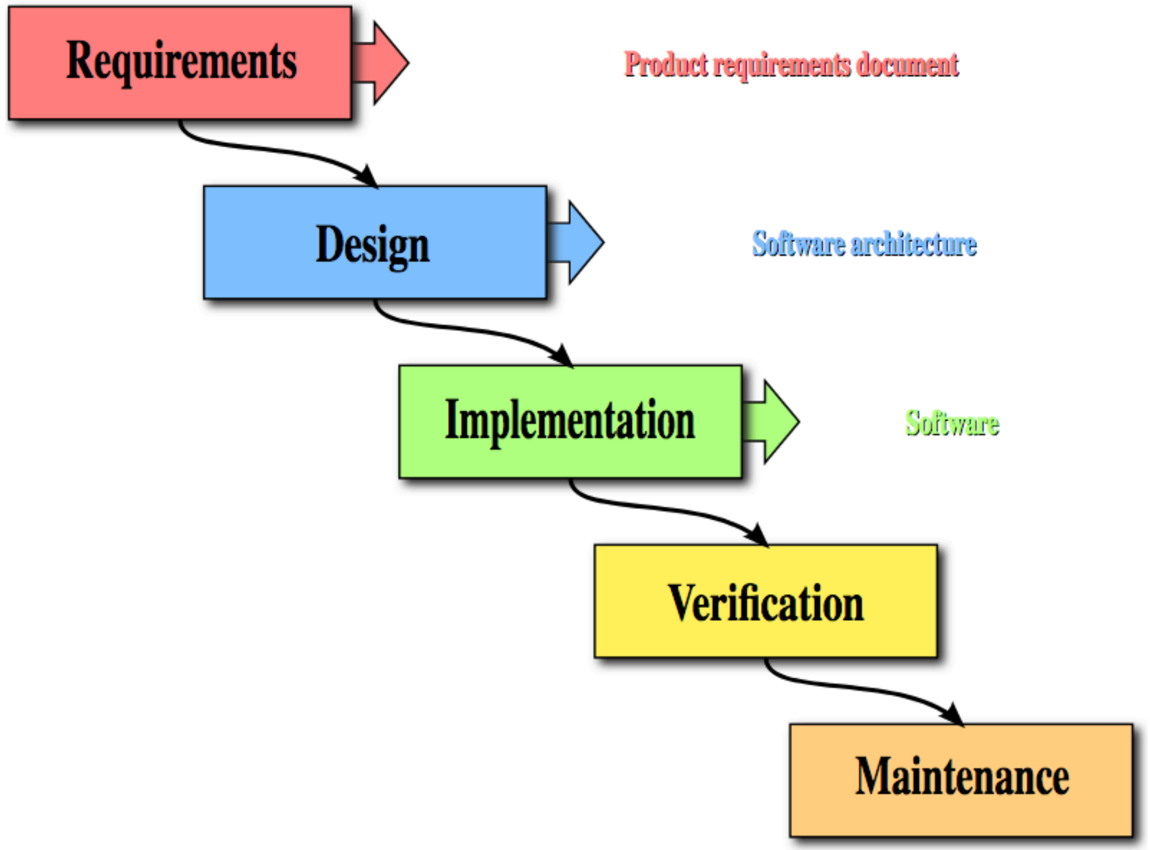
- Waterfall tends to be brittle.
- It is called **heavyweight** because there is no flexibility in the order.

Agile software development methodologies mix up the phases. Agile is characterized by:

- Extensive interactions with the customer.
- Self organizing teams of developers.
- **Iterative development**: frequent production of the working version of the system with new features added for each iteration.
- Quick responses to change.

Waterfall

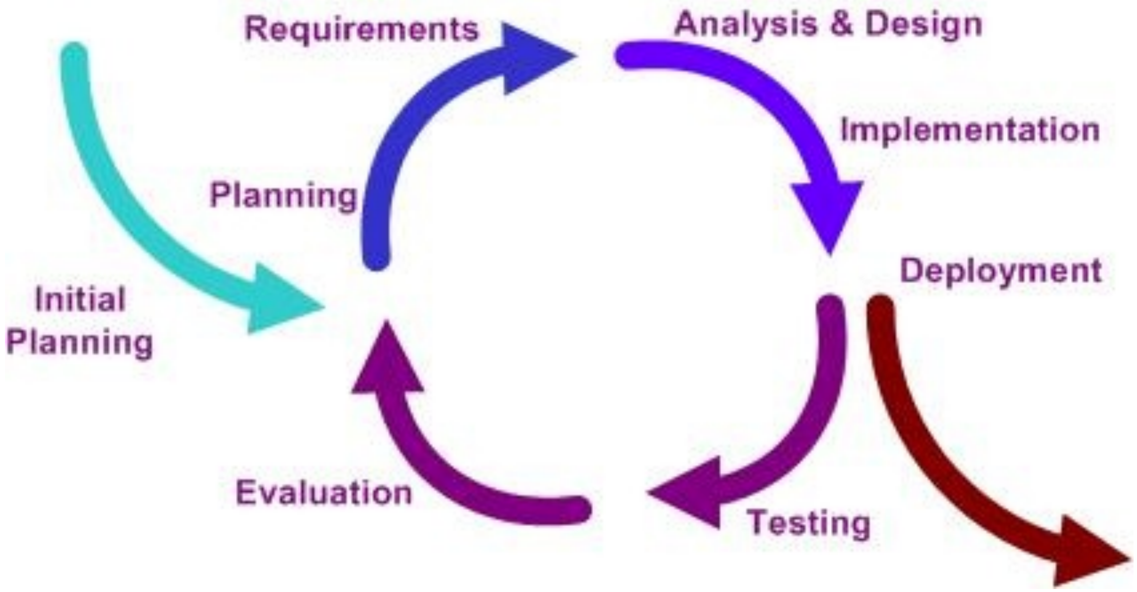
Wikipedia picture – without the feedback arrows.



By Peter Kemp / Paul Smith (Adapted from Paul Smith's work at wikipedia)
[CC BY 3.0 (http://creativecommons.org/licenses/by/3.0)], via Wikimedia Commons

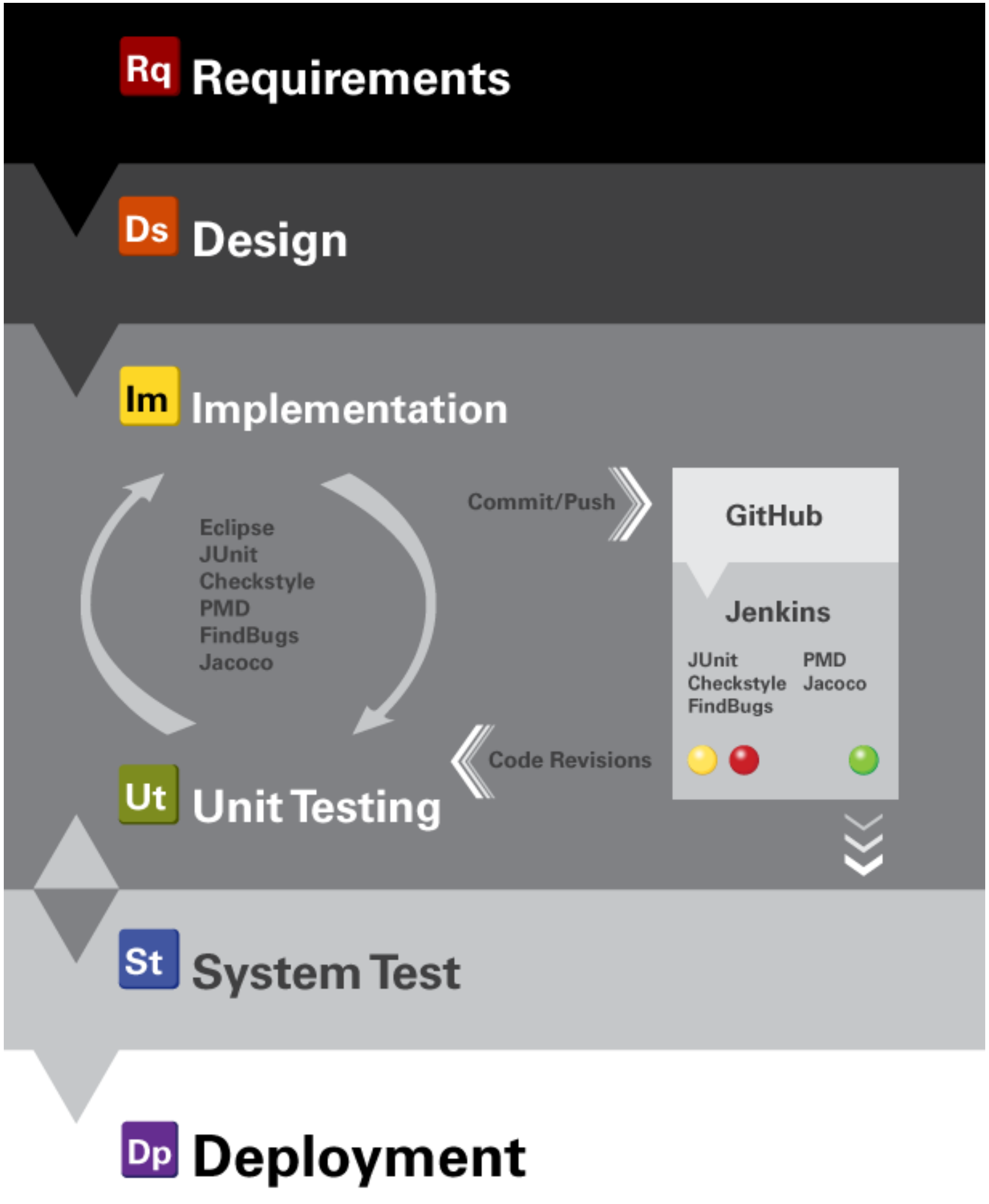
Agile – Iterative Development

Feedback after every part iteration. You get an early working project. You risk scope/feature creep.



By Aflafla1 - Iterative development model V2.jpg , User:Westerhoff, CC0, https://commons.wikimedia.org/w/index.php?curid=34159246

CSC 216 lifecycle



Manifesto for Agile Software Development

Written by 17 software professionals meeting in 2000 to discuss agile software development.

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- ***Individuals and interactions** over processes and tools*
- ***Working software** over comprehensive documentation*
- ***Customer collaboration** over contract negotiation*
- ***Responding to change** over following a plan*

That is, while there is value in the items on the right, we value the items on the left more.

Agile methodologies

Agile methodologies are **lightweight**. Popular ones include

- **XP**. eXtreme Programming. Kent Beck. The first highly publicized agile methodology.
  - Early unit testing
  - Paired programming
- **SCRUM**. Development in 30-day sprints. Takaeuchi, Nagora.
  - Daily 15 minute developer stand-up meeting
  - Code developed in sprints
- **Crystal Clear**. Alistair Cockburn
  - Frequent delivery of useful code to customers
  - Co-location of programmers
- **FDD**. Feature Driven Development. Jeff DeLuca & Peter Coad.
  - Plan, design, and build by feature
- **TDD**. Test Driven Development.
  - Very short development cycles, initiated by test cases.

Best practices

The purposes of best practices include:

- Increased programmer productivity.
- Reduction of software bugs and failures.
- To support collaboration.
- To support the development of high quality software.

There are best practices to support every phase of software development. They:

- Can be used in waterfall and agile/iterative process models.
- Can be supported by tooling.

Some familiar best practices:

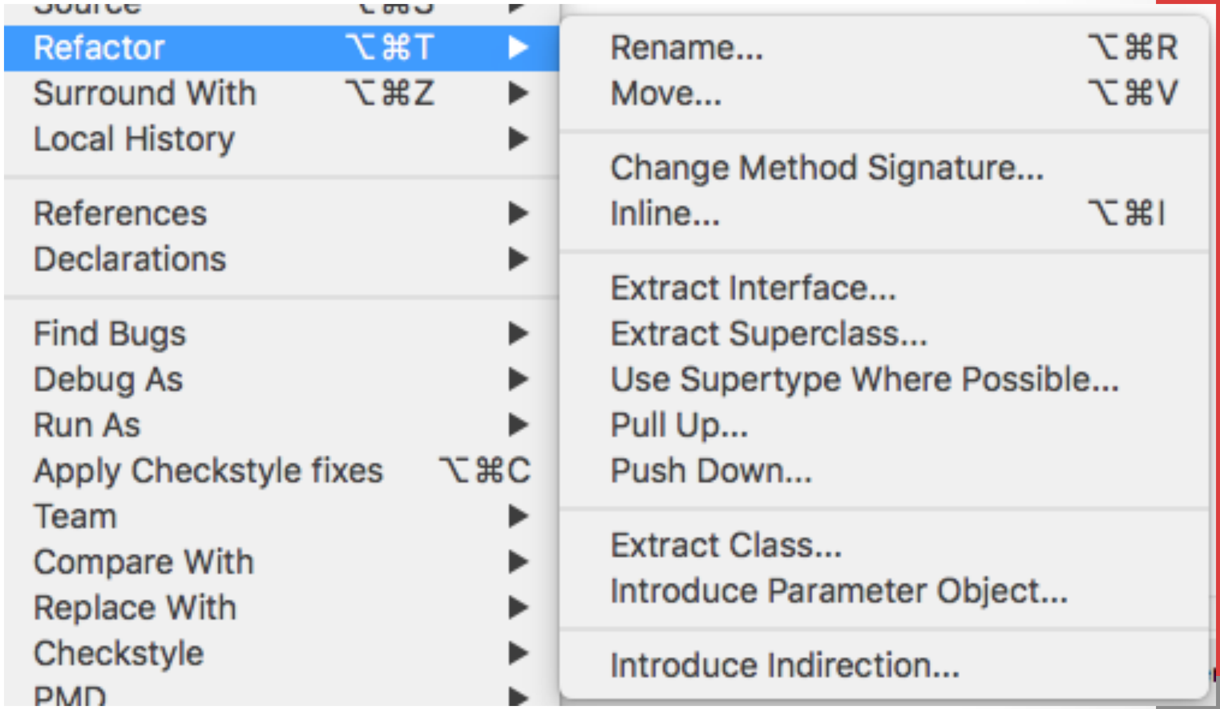
- Using IDEs
- Static Analysis
- Testing Practices
- Version Control (such as Git to keep older and alternate versions)
- Continuous integration (integrating your code into a shared repository)

Best Practice: Refactoring

Refactoring means rewriting code without the intent of changing logic.

Refactoring includes:

- Renaming classes, packages, methods, variables
- Avoiding code duplication by factoring out common code.



Why Refactor?

Reasons to refactor:

- Make code clean and concise.
- Make code easier to understand, modify, and extend.
- Helps avoid duplicate code.
- Helps avoid long methods.

Avoid Long Methods. They almost always are incorrect. They're usually difficult to test because there are often multiple paths through the code.

Perry rule of thumb: Every method with more than 10 lines of code is suspect.

Team development tools

- Software development is virtually always done in teams of two or more. In order not to have code conflicts, development teams use **version control systems**.
- A VCS consists of a repository of files and software that enables you to:
    - check out a file to work on, read, or have as part of your version of the system so far.
    - check in or commit a file after it has been worked on.
    - update the copies of the project files on your own workspace.
    - view a history of changes in a file.
    - roll back a file to a previous version.

References

- Sarah Heckman, CSC 216 slides