Software Development: The Waterfall Model

Steven Zeil

June 7, 2013

Contents

1 Software Development Process Models

Software Development Process Models

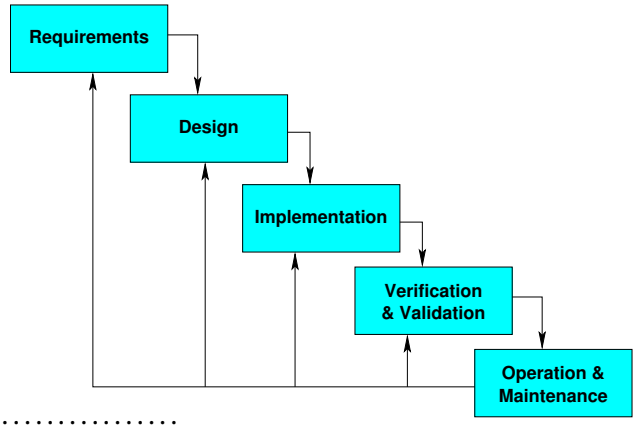
A software development process is a structured series of activities that comprise the way in which an organization develops software projects.

- Most common: the Waterfall model
- Evolutionary, Spiral, & Unified models
  - outside the scope of this course

. . . . . .

The Waterfall Model

- Movement is always forward from phase to phase
- Management often must sign off on deliverables from each phase
  - advantage: accountability
  - disadvantage: inflexibility



. . . . . .

1.1 Components of the Waterfall Model

Requirements analysis and definition

- Establishing what the customer requires from a software system

1.1.1 What is a requirement?

- A statement of something the system must do
  - external behaviors
- or of a constraint under which the system must work

May range from very abstract to a detailed mathematical specification

. . . . . .

The requirements document

- The requirements document is the official statement of what is required of the system developers
- It is not a design document. As far as possible, it should state *WHAT* the system should do rather than *HOW* it should do it
- Should include both a definition and a specification of requirements

. . . . . .

Requirements definition

- *Requirements definition*: A statement in natural language plus diagrams of the services the system provides and its operational constraints.
  - Written for customers

Example
1. The software must provide a means of representing and accessing external files created by other tools.

. . . . . .

Requirements specification

- Requirements specification: A structured document setting out detailed descriptions of the system services.
  - Written as a contract between client and contractor

Sample Reqts Specification
1.1 The user should be provided with facilities to define the type of external files
1.2 Each external file type may have an associated tool which may be applied to the file.
1.3 Each external file type may be represented as a specification (icon) on the user's display.
1.4 Facilities should be provided for the icon representing an external file type to be defined by the user.
1.5 When the user selects an icon representing an external file, the effect of that selection is to apply the tool associated with the type of the external file to the file represented by the selected icon.

. . . . . .

System and software design

- Deriving a solution which satisfies software requirements
- Design can occur in multiple phases

  Architectural design global decisions that affect the entire system

  High-Level Design Divide the system into modules

  Low-Level Design Design the data structures and algorithms for an individual module.

. . . . . .

Architectural Design

Global decisions that affect the entire system.
Examples:

- Should we use regular files or have a database?
- Does this run on one computer, or is it a distributed system with many cooperating computers?
- How will run-time errors be handled?

. . . . . .

High-Level Design

Divide the system into modules

- Guided by
  - coupling & cohesion
  - abstract data types (later this semester)
  - objects (studied in CS330)

. . . . . .

Low-Level Design

Design the data structures and algorithms for an individual module.

- Guided by
  - stepwise refinement (later this semester)
  - analysis of algorithms (studied in CS361 & CS390)

. . . . . .

Implementation

Probably the most familiar activity

- programming

. . . . . .

Verification & Validation

Verification & Validation: assuring that a software system meets the users' needs

- principal objectives:
  - The discovery of defects in a system
  - The assessment of whether or not the system is usable

. . . . . .

Verification

- *Verification*:
  - "Are we building the product right?"
  - The software should conform to its prior specification
    - the code should be faithful to the low-level design
    - the low-level design should be faithful to the high-level design
    - the high-level design should be faithful to the requirements specification

. . . . . .

Validation

- *Validation*:
  - "Are we building the right product?"
  - The software should do what the user really requires

1.1.2 Testing

- *Testing* is the act of executing a program with selected data to uncover bugs.
  - As opposed to *debugging*, which is the process of finding the faulty code responsible for failed tests.
- Testing is the most common, but not the only form of V&V

. . . . . .

V&V

- Often portrayed as final phase of waterfall
- Is actually a full-life-cycle activity
  - Requirements must be validated
  - Designs may be validated & verified
  - Implementation is tested
  - final system is tested
  - maintenance changes are tested

. . . . . .

Operation and maintenance

As requirements evolve and bug reports come in from the field

- prioritize changes
- make changes (a kind of mini-s.d.p.)
- validate changes
  - new test cases
- validate that change does not break previously working code
  - regression testing

. . . . . .

2 The Focus of CS250

The Focus of CS250