Software Process Models

Topics
- software process
- software process phases
- waterfall models
- V-models

How do you build software?
There are four generic activities or phases:
- specification: identifying the required services and the constraints on system operation & development
- development: converting the system specification into an executable system
- validation: showing that a system conforms to its specification and meets the user requirements
- evolution: adapting software to changes in circumstances

Software processes
Different software processes are possible that differ in:
- how the phases are arranged
- how much attention is paid to each phase
  - whether and how the phases are broken down
- what must be done in each phase
- when and how to transition between phases
  - what must be delivered by each phase

"Null process:"
- development
- validation
  - "code and fix"
**Specification**

Main concern: **what** should the system do?

Main result: **requirements** == description of desired system functionality

- **user requirements**: abstract, natural-language
  - aimed at users, customer, and contractor
- **system requirements**: specific, semi-formal
  - aimed at developers and contractor

**Requirements engineering tasks**

- **feasibility study**: check whether a new system is needed, and whether it can be build it within the given time and budget
- **requirements elicitation**: gather requirements by
  - user interviews and questionnaires
  - user observation (ethnographic studies)
  - brainstorming and role playing
  - use cases and prototyping.
- **requirements analysis**: classify, organize, and prioritize requirements
  - can prompt more elicitation

**Requirements engineering is hard...**

**Development**

**development**: converting the system specification into an executable system

Main concern: **how** should the system work?

**Development = Design + Implementation**

**development**: converting the system specification into an executable system

Traditionally broken down into several stages:

- architectural design
- interface design
- abstract specification
- coding
- **development** is an iterative process with feedback between the stages
- design and implementation are typically interleaved
**Validation**

**validation**: showing that a system conforms to its specification and meets the user requirements

Main concern: **does the system work?**

---

**Verification and Validation**

**validation**: showing that a system conforms to its specification and meets the user requirements

Traditionally broken down into two separate activities:
- **verification**: do we build the system right?
- **validation**: do we build the right system?

V&V techniques:
- reviews (code and documents)
- prototyping and simulation (documents)
- **testing**
- static analysis (e.g., model checking, proof)

---

**Evolution**

**evolution**: adapting software to changes in circumstances

Main concern: **keep the system working**

Traditionally broken down into several categories:
- **corrective**: fixing bugs in the field
- **adaptive**: respond to changes in environment
- **perfective**: respond to changes in user requirements
- **preventive**: improve or refactor system to prevent future problems

---

**Distribution of efforts**

Verification effort dominates both specification and development efforts!

---

... but maintenance (50-75%) dominates the total life cycle effort!
Distribution of maintenance efforts

- perfective: 50%
- adaptive: 25%
- corrective: 21%
- preventive: 4%


Requirements-driven changes dominate maintenance efforts!

Software process modeling

A software process model is a generic description of a class of software processes.

- typically described informally
  - “manifestos”, books, standards
  - case studies
- can also be formalized / implemented
  - foundation for project management systems
- we will identify software process and software process model

Source:

Dimensions of software processes

- linear vs. iterative
  - phases are executed in order
  - phases are repeated regularly
- monolithic vs. incremental
  - single system is delivered at end
  - system is delivered in several working versions with increasing functionality
- document-based vs. code-based
  - most deliverables (except final system) are text
  - almost all deliverables are code (prototypes, test cases, ...)

Dimensions of software processes

- heavy-weight vs. light-weight
  - strict rules and detailed plans, formal models and methods
  - flexible and reactive, prototyping
  - also called plan-driven and agile
- component-driven vs. model-driven vs. test-driven
  - requirements are modified to fit existing components
  - requirements are formulated as (formal) models
  - requirements are formulated as test cases

Note: not all combinations possible or sensible...

Code-and-fix

- No specification, no design
- Just dive in and starting coding

Advantages
- no overhead
- see progress quickly
- suitable for small, short-lived programs (<200 lines)

Disadvantages
- no way to assess progress, quality, or risks
- changes likely to require major design overhaul

Classic waterfall model

- each phase must be completed before next is started
- each phase transition is marked by delivery of corresponding documents

More information:
**Classic waterfall model: Advantages**

- phase distinction and order encourages discipline
  - can provide support for inexperienced development teams
- emphasis on communication through documents
  - promotes documentation
  - support for off-shoring
  - allows automation (if documents are formal models)
- implementation postponed until project objectives are well-understood
  - minimizes risk of waste

**Classic waterfall model: Disadvantages**

- inflexible partitioning of project into distinct stages
  - no feedback, but specification problems often show only during coding or integration
  - validation too late
- first working version available only very late
  - little feedback from users until then
  - increases risk of obsolescence
- requires accurate and complete requirements right from the start
  - difficult to respond to changing requirements
- over-reliance on documents, heavy-weight process
  - documents get out-of-sync

---

**The classic waterfall model is a strawman.**

- Phases never meant to be followed strictly linearly!!
- "...the implementation described above is risky and invites failure."

---

**Modified waterfall model**

- previous phase can be revisited when problems occur
  - increases flexibility
- ...but requires more synchronization of documents
  - ...and often more than one earlier phase is affected
  - ...and validation is still late

**V-model: mirror all specification and development steps by validation steps.**

- basically still a waterfall model!
Double-Vee: use and test models.

Triple-Vee: verify tests.

Big Design Up-Front models (BDUF)

BDUF denotes any model where the design is completed and perfected before the implementation is started.

- derived from waterfall model
- suitable for
  - well-understood domains with stable requirements
  - high-reliability systems
- unsuitable for
  - highly complex systems in ill-understood domains
  - changing requirements

Iterative methods

- spiral model
- rapid application development (RAD) / prototyping
- rational unified process (RUP) / modelling
- agile / lean / scrum development
- eXtreme Programming

We will start with a waterfall model and look at these later...

Summary

- software development process is split into phases:
  - specification − development
  - validation − maintenance
- maintenance dominates total life cycle effort
- software process models describe how the phases are arranged and what must be done in each phase
- main distinction: linear vs. iterative
- the linear waterfall model is the traditional model
  - good for stable requirements, bad for change