CS 410/510 - Software Engineering

Software Processes

Reference: Sommerville, Software Engineering, 10 ed., Chapter 2

The big picture

A software process is a structured set of activities required to develop a software system. Note that we are talking about a "software process" -- not a "software *development* process."

There are many different kinds of software processes, but each and every one of them involve these four types of fundamental activities:

- Software **specification** - defining what the system should do;
- Software **design and implementation** - defining the organization of the system and implementing the system;
- Software **validation** - checking that it does what the customer wants;
- Software **evolution** - changing the system in response to changing customer needs.

A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective. When we describe and discuss software processes, we usually talk about the activities in these processes such as specifying a data model, designing a user interface, etc. and the ordering of these activities. Process descriptions may also include:

- **Products**, which are the outcomes of a process activity;
- **Roles**, which reflect the responsibilities of the people involved in the process;
- Pre- and post-**conditions**, which are statements that are true before and after a process activity has been enacted or a product produced.

Plan-driven processes are processes where all of the process activities are planned in advance and progress is measured against this plan. In **agile** processes, planning is incremental and it is easier to change the process to reflect changing customer requirements. In practice, most practical processes include elements of both plan-driven and agile approaches.

Software process models

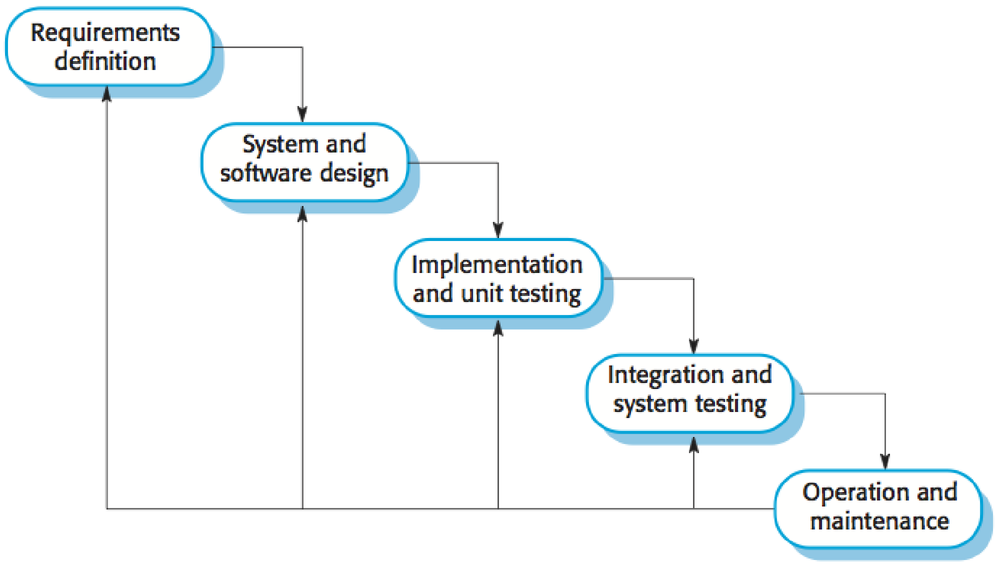
The waterfall model
Plan-driven model. Separate and distinct phases of specification, software design, implementation, testing, and maintenance.

Incremental development
Specification, development and validation are interleaved. The system is developed as a series of versions (increments), with each version adding functionality to the previous version. May be plan-driven or agile.

Integration and configuration
Based on the existence of a significant number of reusable components/systems. The system development process focuses on integrating these components into a system rather than developing them from scratch. May be plan-driven or agile.

In practice, most large systems are developed using a process that incorporates elements from all of these models.

The waterfall model


Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

There are separate identified **phases** in the waterfall model:

Requirements analysis and definition
: The system's services, constraints, and goals are established by consultation with system users. They are then defined in detail and serve as a system specification.

System and software design
: The systems design process allocates the requirements to either hardware or software systems by establishing an overall system architecture. Software design involves identifying and describing the fundamental software system abstractions and their relationships.

Implementation and unit testing
: During this stage, the software design is realized as a set of programs or program units. Unit testing involves verifying that each unit meets its specification.

Integration and system testing
: The individual program units or programs are integrated and tested as a complete system to ensure that the software requirements have been met. After testing, the software system is delivered to the customer.

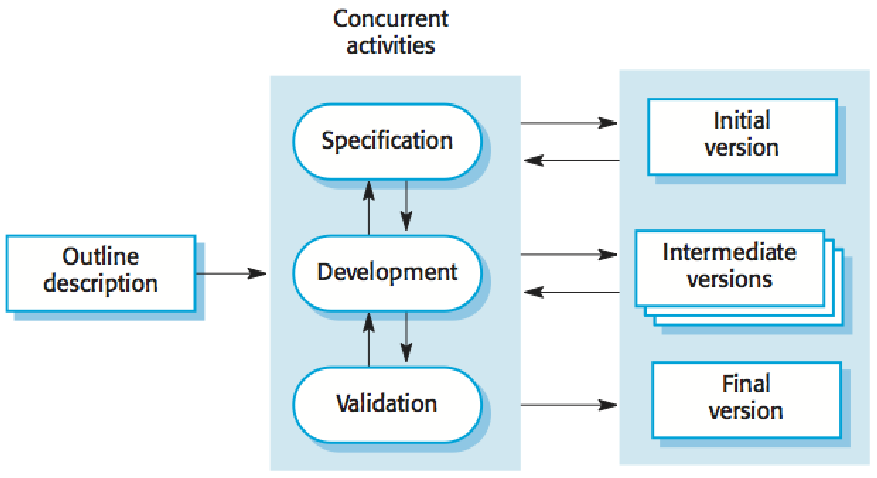
Operation and maintenance
: Normally (although not necessarily), this is the longest life cycle phase. The system is installed and put into practical use. Maintenance involves correcting errors which were not discovered in earlier stages of the life cycle, improving the implementation of system units and enhancing the system's services as new requirements are discovered.

The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway. In principle, a phase has to be complete before moving onto the next phase. Waterfall model **problems** include:

Difficult to address change
: Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements. Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process. Few business systems have stable requirements.

Very few real-world applications
: The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites. In those circumstances, the plan-driven nature of the waterfall model helps coordinate the work.

Incremental development model



Benefits of incremental development:

Lower cost of changes
The cost of accommodating changing customer requirements is reduced. The amount of analysis and documentation that has to be redone is much less than is required with the waterfall model.

Frequent feedback
It is easier to get customer feedback on the development work that has been done. Customers can comment on demonstrations of the software and see how much has been implemented.

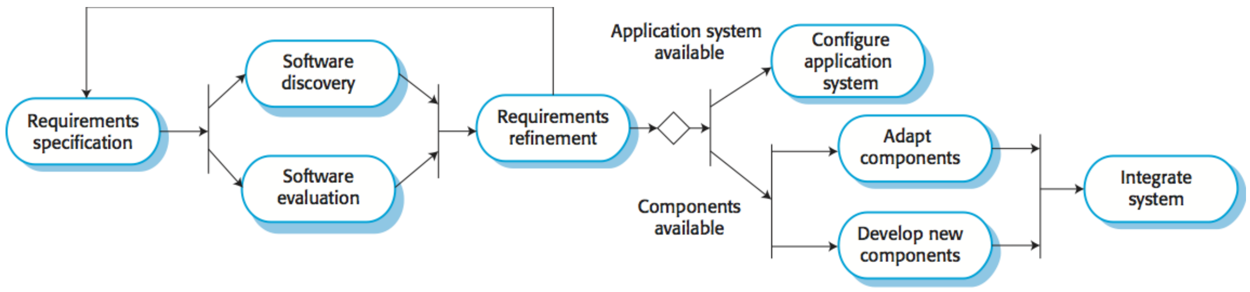
Faster delivery
More rapid delivery and deployment of useful software to the customer is possible. Customers are able to use and gain value from the software earlier than is possible with a waterfall process.

Problems with incremental development (from the management perspective):

The process is not visible
Managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of the system.

System structure tends to degrade as new increments are added
Unless time and money is spent on refactoring to improve the software, regular change tends to corrupt its structure. Incorporating further software changes becomes increasingly difficult and costly.

Integration and configuration



This approach is based on systematic reuse where systems are integrated from existing components or COTS (Commercial-off-the-shelf) systems. Process stages include:

- Component analysis;
- Requirements modification;
- System design with reuse;
- Development and integration.

Reuse is now the standard approach for building many types of business system.

Types of software components:

- **Web services** that are developed according to service standards and which are available for remote invocation.
- Collections of objects that are developed as a **package** to be integrated with a component framework such as .NET or J2EE.
- Stand-alone **commercial-off-the-shelf** systems (COTS) that are configured for use in a particular environment.

Software process activities

Real software processes are inter-leaved sequences of technical, collaborative and managerial activities with the overall goal of specifying, designing, implementing and testing a software system.

The four basic process activities of specification, development, validation and evolution are organized differently in different development processes. In the waterfall model, they are organized in sequence, whereas in incremental development they are interleaved.

Software specification

The process of establishing what services are required and the constraints on the systemâ€™s operation and development.

Requirements engineering process:

- **Feasibility study**: is it technically and financially feasible to build the system?
- Requirements **elicitation and analysis**: what do the system stakeholders require or expect from the system?
- Requirements **specification**: defining the requirements in detail
- Requirements **validation**: checking the validity of the requirements

Software design and implementation

The process of converting the system specification into an executable system.

- **Software design**: design a software structure that realizes the specification;
- **Implementation**: translate this structure into an executable program;

The activities of design and implementation are closely related and may be interleaved.

Design activities include:

- **Architectural design**: identify the overall structure of the system, the principal components (sometimes called sub-systems or modules), their relationships and how they are distributed.
- **Interface design**: define the interfaces between system components.
- **Component design**: take each system component and design how it will operate.
- **Database design**: design the system data structures and how these are to be represented in a database.

Software validation

Verification and validation (V & V) is intended to show that a system conforms to its specification and meets the requirements of the system customer.

- **Validation**: are we building the right product (what the customer wants)?
- **Verification**: are we building the product right?

V & V involves checking and review processes and system testing. System testing involves executing the system with test cases that are derived from the specification of the real data to be processed by the system.

Testing is the most commonly used V & V activity and includes the following stages:

- **Development or component testing**: individual components are tested independently; components may be functions or objects or coherent groupings of these entities.
- **System testing**: testing of the system as a whole, testing of emergent properties is particularly important.

- **Acceptance testing**: testing with customer data to check that the system meets the customer's needs.

Software evolution
: Software is inherently flexible and can change. As requirements change through changing business circumstances, the software that supports the business must also evolve and change. Although there has been a demarcation between development and evolution (maintenance) this is increasingly irrelevant as fewer and fewer systems are completely new.

Coping with change

Change is inevitable in all large software projects. Business changes lead to new and changed system requirements New technologies open up new possibilities for improving implementations Changing platforms require application changes Change leads to rework so the costs of change include both rework (e.g. re-analyzing requirements) as well as the costs of implementing new functionality.

Two strategies to **reduce the costs** of rework:

Change avoidance
: The software process includes activities that can anticipate possible changes before significant rework is required. For example, a prototype system may be developed to show some key features of the system to customers.

Change tolerance
: The process is designed so that changes can be accommodated at relatively low cost. This normally involves some form of incremental development. Proposed changes may be implemented in increments that have not yet been developed. If this is impossible, then only a single increment (a small part of the system) may have be altered to incorporate the change.

Software prototyping

A prototype is an initial version of a system used to demonstrate concepts and try out design options. A prototype can be used in:

- The requirements engineering process to help with requirements elicitation and validation;
- In design processes to explore options and develop a UI design;
- In the testing process to run back-to-back tests.

Benefits of prototyping:

- Improved system usability.
- A closer match to users' real needs.
- Improved design quality.
- Improved maintainability.
- Reduced development effort.

Prototypes may be based on rapid prototyping languages or tools. They may involve **leaving out functionality**:

- Prototype should focus on areas of the product that are not well-understood;
- Error checking and recovery may not be included in the prototype;
- Focus on functional rather than non-functional requirements such as reliability and security.

Prototypes should be **discarded** after development as they are not a good basis for a production system:

- It may be impossible to tune the system to meet non-functional requirements;
- Prototypes are normally undocumented;
- The prototype structure is usually degraded through rapid change;
- The prototype probably will not meet normal organizational quality standards.

Incremental development/delivery

Rather than deliver the system as a single delivery, the development and delivery is broken down into increments with each increment delivering part of the required functionality. User requirements are prioritized and the highest priority requirements are included in early increments. Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve.

Advantages of incremental delivery:

- Customer value can be delivered with each increment so system functionality is available earlier.
- Early increments act as a prototype to help elicit requirements for later increments.
- Lower risk of overall project failure.
- The highest priority system services tend to receive the most testing.

Incremental delivery **problems**:

- Most systems require a set of basic facilities that are used by different parts of the system. As requirements are not defined in detail until an increment is to be implemented, it can be hard to identify common facilities that are needed by all increments.
- The essence of iterative processes is that the specification is developed in conjunction with the software. However, this conflicts with the procurement model of many organizations, where the complete system specification is part of the system development contract.

Process improvement

Many software companies have turned to software process improvement as a way of enhancing the quality of their software, reducing costs or accelerating their development processes. Process improvement means understanding existing processes and changing these processes to increase product quality and/or reduce costs and development time.

Process maturity approach
Focuses on improving process and project management and introducing good software engineering practice. The level of process maturity reflects the extent to which good technical and management practice has been adopted in organizational software development processes.

Agile approach
Focuses on iterative development and the reduction of overheads in the software process. The primary characteristics of agile methods are rapid delivery of functionality and responsiveness to changing customer requirements.

Process improvement activities form a continuous cycle with a feedback loop:

- **Measure** one or more attributes of the software process or product. These measurements forms a baseline that help decide if process improvements have been effective.
- **Analyze** the current process and identify any bottlenecks.
- **Change** the process to address some of the identified process weaknesses. These are introduced and the cycle resumes to collect data about the effectiveness of the changes.

Process measurement

- Wherever possible, quantitative process data should be collected.
- Process measurements should be used to assess process improvements.
- Metrics may include:
  - Time taken for process activities to be completed, e.g. calendar time or effort to complete an activity or process.
  - Resources required for processes or activities, e.g. total effort in person-days.
  - Number of occurrences of a particular event, e.g. number of defects discovered.

The SEI capability maturity model

- **Initial:** Essentially uncontrolled
- **Repeatable:** Product management procedures defined and used
- **Defined:** Process management procedures and strategies defined and used
- **Managed:** Quality management strategies defined and used
- **Optimizing:** Process improvement strategies defined and used