SOFTWARE ENGINEERING

Week 2

Software Processes and Process Models

Prof. Dr. Muhittin GÖKMEN Yard. Doç. Dr. A. Cüneyd TANTUĞ Araş. Gör. Dr. Tolga OVATMAN
Istanbul Technical University
Computer Engineering Department

Agenda

1. Software Processes
2. Plan Driven Software Process Models

1. Software Processes
2. Plan Driven Software Process Models

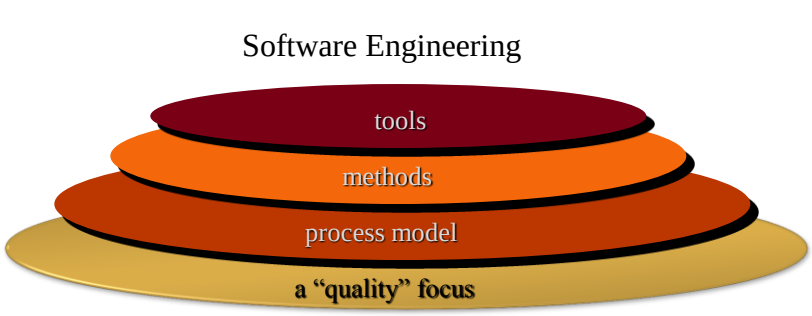
Software Processes

2.1

The Software Process

- A structured set of activities (framework) required to develop a software system.
- Many different software processes exists, but all involve:
  - Specification – defining what the system should do;
  - Design and implementation – defining the organization of the system and implementing the system;
  - Validation – checking that it does what the customer wants;
  - 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.

A Layered Technology



Software Engineering Primary Activities

Framework Activities

- Planning
- Modeling
  - Analysis of requirements
  - Design of modules and interfaces
- Construction
  - Coding
  - Testing
- Deployment

Software Engineering Secondary Activities

Umbrella Activities

- Software project management
- Formal technical reviews
- Software quality assurance
- Software configuration management
- Work product preparation and production
- Reusability management
- Measurement
- Risk management

Plan-Driven and Agile Processes

Plan-Driven Processes

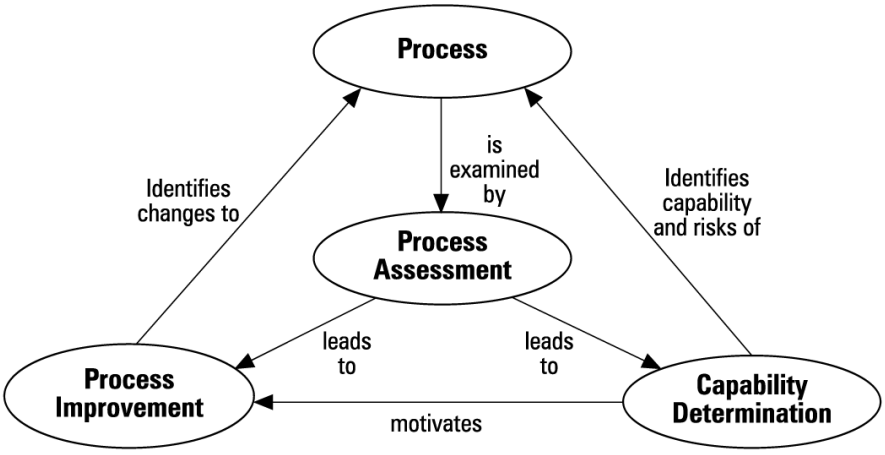
- Plan-driven processes are processes where all of the process activities are planned in advance and progress is measured against this plan.

Agile Processes

- 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.
- There are no right or wrong software processes.

Process Assesment and Improvement



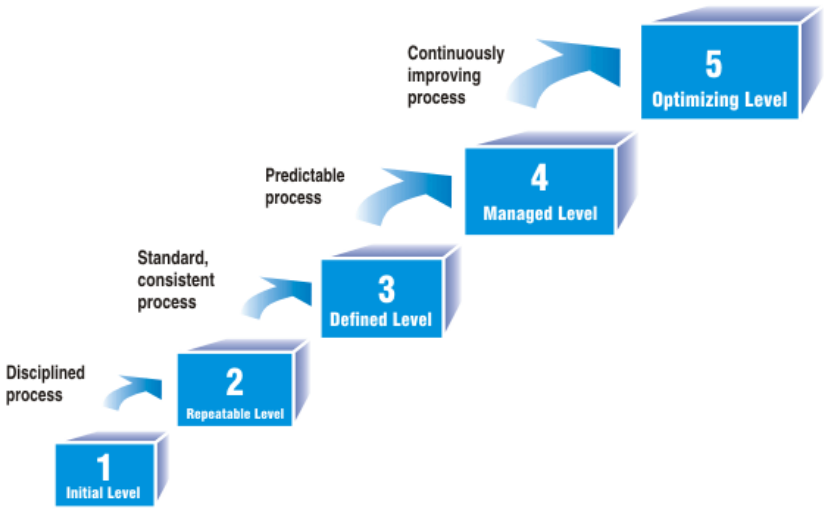
Process Assesment & Improvement

- The process should be assessed to ensure that it meets a set of basic process criteria that have been shown to be essential for a successful software engineering.
- Many different assessment options are available:
  - CMMI (Capability Maturity Model Integration)
    Independent assessments grade organizations on how well they follow their defined processes, **not** on the quality of those processes or the software produced
  - ISO 9001:2000
    Certification with ISO 9000 does not guarantee the quality of the end result, only that formalized business processes have been followed.
  - SPICE-Software Process Improvement Capability dEtermination (ISO15504)
  - SCAMPI
  - CBA IPI

The CMMI

- CMM is an integrated capability model that includes software and systems engineering capability assessment.
- The Software Engineering Institute (SEI) published the revised framework CMMI in 2001.
- Process capability assessment
  - Intended as a means to assess the extent to which an organisation's (such as a software company) processes follow best practice.

CMM Levels



CMM Level Definitions

1. Initial
   - Essentially uncontrolled
   - Ad-hoc people management
2. Repeatable
   - Product management procedures defined and used
   - Policies developed for capability improvement
3. Defined
   - Process management procedures and strategies defined and used
   - Standardised people management across the organisation
4. Managed
   - Quality management strategies defined and used
   - Quantitative goals for people management in place
5. Optimising
   - Process improvement strategies defined and used
   - Continuous focus on improving individual competence

Personal Software Process (PSP)

- Recommends five framework activities:
  - Planning
  - High-level design
  - High-level design review
  - Development
  - Postmortem
- stresses the need for each software engineer to identify errors early and as important, to understand the types of errors

Team Software Process (TSP)

- Each project is “launched” using a “script” that defines the tasks to be accomplished
- Teams are self-directed
- Measurement is encouraged
- Measures are analyzed with the intent of improving the team process

Plan Driven Software Process Models

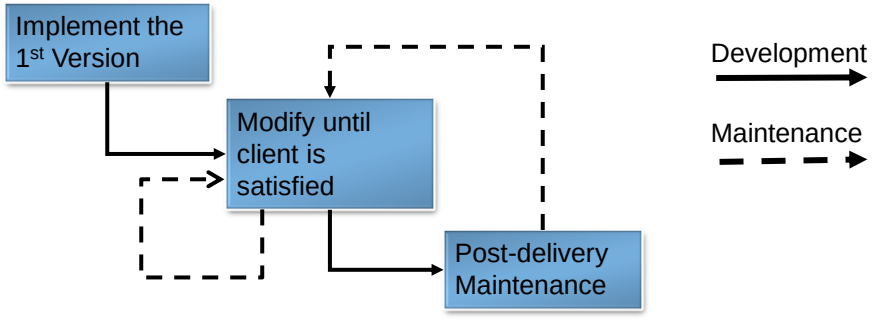
2.2

Plan-Driven Software Process Models

1. Code-and-fix model
2. Waterfall model
3. Incremental model
4. Rapid prototyping model
5. Iterative model
6. Unified Process model
7. Component-Based model

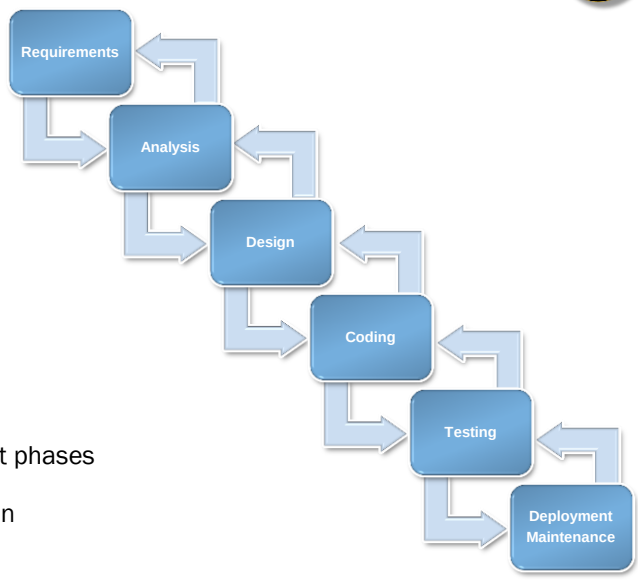
- In practice, most large systems are developed using a process that incorporates elements from all of these models.

1. Code-and-Fix Model



- The easiest way to develop software
- No design, no specifications
- Maintenance extremely difficult
- The most expensive way
- Typically used by a start-up

2. Waterfall Model



- Separate and distinct phases
- Feedback loops
- Documentation-driven

2. Waterfall Model Pros and Cons

Pros

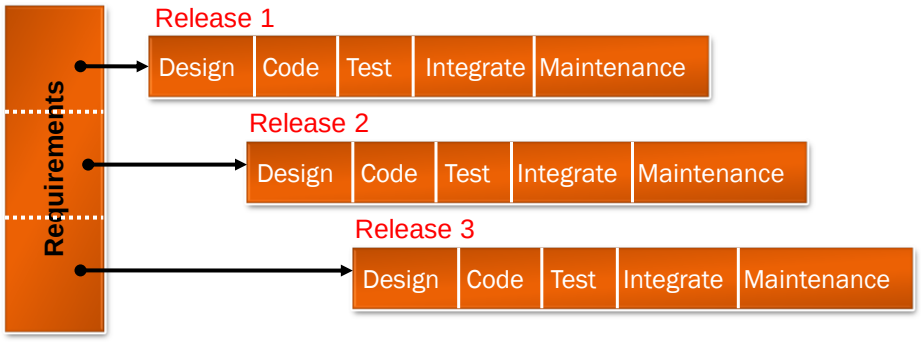
- Simple and disciplined, structured approach
- Project Management is easy
- Maintenance is easier

- This model is only appropriate when the requirements are well-understood and changes will be limited during the design process.
- 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.

Cons

- Difficulty of accommodating change after the process is underway.
- Necessitates stable requirement
  - Few business systems have stable requirements.
    - Military, Government Projects
- Major design problems may not be detected till very late.
- Very late delivery
- Blocking phases
  - Coders must wait designers to prepare design document

3. Iterative/Incremental Development



- Avoids "big bang" implementation
- Assumes all requirements known up-front
- Each release adds more functionality
- Once the development of an increment is started, the requirements are **frozen** though requirements for later increments can continue to evolve.

3. Incremental Development Pros and Cons

Pros

- 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.
- 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.
- 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.

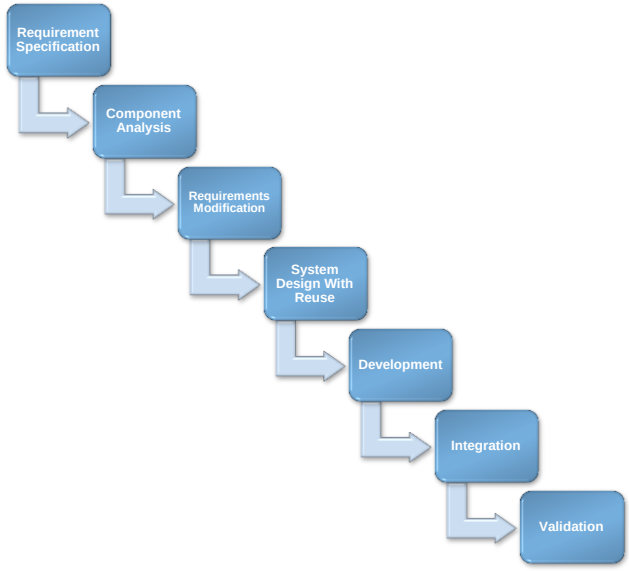
Cons

- 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.

4. Component Based Development

- Based on systematic reuse where systems are integrated from existing components or COTS (Commercial-off-the-shelf) systems.
- Process stages
  - Component analysis
  - Requirements modification
  - System design with reuse
  - Development and integration
- Reuse is now the standard approach for building many types of business system
- The followings are examples of component standards which have their own component libraries and consistent structures.
  - OMG / CORBA
  - Microsoft COM
  - Sun JavaBeans

4. Component Based Development



4. Component Based Development Pros & Cons

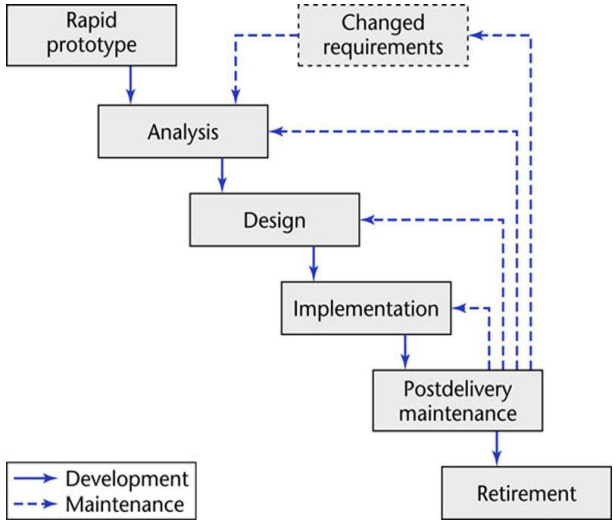
Pros

- system reliability is increased (standard reusable components should be well tested and perhaps formally verified)
- development time is reduced
  - design and coding time is reduced
  - testing time is reduced
- standards can be implemented as reusable components
  - standards for fault-tolerance or correctness
  - standards for user interfaces
    - a company's "look and feel" could come from reuse of standard user interface components

Cons

- reusing components leads to changes in design and requirements
- developers always believe they could develop better components anyway
- incorporating component libraries often leads to larger and less efficient implementations
- organizations are reluctant to expend resources ($$) to develop reusable components
- no standard way to catalog and search for reusable components
- no guarantee that reusing components leads to faster development or more reliable systems
- new versions of purchased components are not controlled by the development organization, which may affect system evolution

5. Rapid Prototyping Model



5. Rapid Prototyping Model

- Prototyping is used for:
  - understanding the requirements for the user interface
  - can start with initial requirements to clarify what is really needed
  - examining feasibility of a proposed design approach
  - exploring system performance issues
- Preferred for new technology projects.
- A prototype has only a limited capability.
- Mostly prototyping takes 3-4 months.

5. Prototype Development and Retirement

- May be based on rapid prototyping languages or tools
- 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.

5. Rapid Prototyping Model Pros and Cons

Pros

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

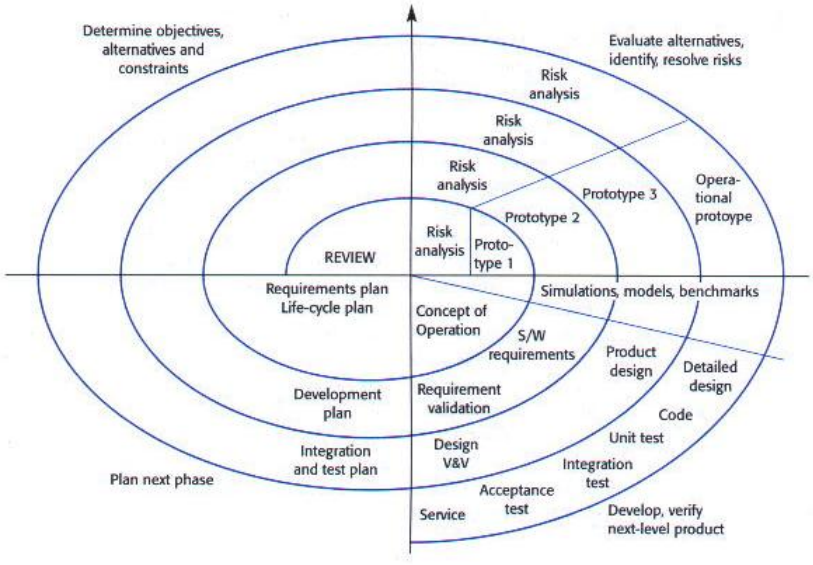
Cons

- Usually the customer insists on «small modifications» to prototype system after seeing sth appears to be a working version of the software.
- The developer may use inappropriate components for building prototype quickly. By time, they get comfortable with the choices and forget all reasons why ther were inappropriate. Less-then-ideal choices become a part of the system.

6. Spiral Model

- The spiral model is a software development process combining the elements of both design and prototyping-in-stages.
- This model of development combines the features of the prototyping model and the waterfall model.
- The spiral model is intended for large, expensive and complicated projects.
- Process is represented as a spiral rather than as a sequence of activities with backtracking.
- Each loop in the spiral represents a phase in the process.
- No fixed phases such as specification or design - loops in the spiral are chosen depending on what is required.
- Risks are explicitly assessed and resolved throughout the process.

6. Spiral Model



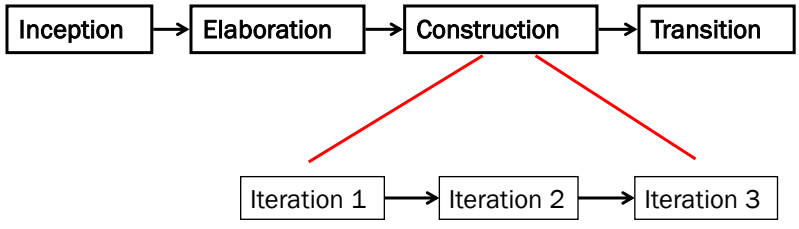
6. Spiral Model Sectors

- Objective setting
  - Specific objectives for the phase are identified.
- Risk assessment and reduction
  - Risks are assessed and activities put in place to reduce the key risks.
- Development and validation
  - A development model for the system is chosen which can be any of the generic models.
- Planning
  - The project is reviewed and the next phase of the spiral is planned.

7. The Unified Process

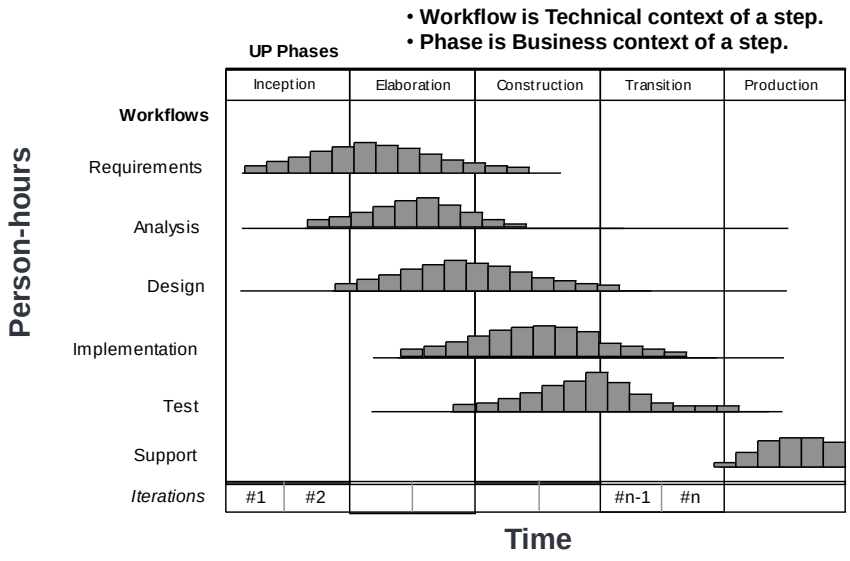
- The Unified Process (UP) is a "use-case driven, iterative and incremental" software process model closely aligned with Object-Oriented Analysis and Design.
- A modern generic process derived from the work on the UML and associated process.
- Brings together aspects of the 3 generic process models discussed previously.
- Normally described from 3 perspectives
  - A dynamic perspective that shows phases over time;
  - A static perspective that shows process activities;
  - A practive perspective that suggests good practice.
- Each phase ends at a major milestone and contains one or more iterations.
- An iteration is a distinct sequence of activities with an established plan and evaluation criteria, resulting in an executable release.

7. The Unified Process - II



- Each iteration is defined in terms of the scenarios it implements.

7. The Unified Process—Workflows&Phases



7. The Unified Process Phases-I

1. Inception

- Establish business rationale for project
- Decide project scope
- Identify actors and use cases

- Work Products (artifacts):
  - Vision document
  - Initial use-case model
  - Initial risk assessment
  - Project plan
  - Prototype

2. Elaboration

- Collect more detailed requirements
- Do high-level analysis and design
- Establish baseline architecture
- Create construction plan

- Work Products:
  - Use-case model
  - Non-functional requirements
  - Analysis model
  - Software architecture description
  - Preliminary design model
  - Preliminary user manual

7. The Unified Process Phases-II

3. Construction

- Build, test and validate the project
- Work Products:
  - Design model
  - Software components
  - Test plan and test cases
  - Support documenatation
    - User manuals
    - Installation manuals
    - Description of current increment

4. Transition

- Beta-test
- Tune performance
- Train users
- Work Products:
  - Delivered software increment
  - Beta test results
  - General user feedback

Other Process Models

- Agile Methodologies
  Will be covered next week.
- Formal Methods Model
  Emphasizes the mathematical specification of requirements
  Will be covered last week in «**Advanced Software Engineering**» topics.
- Aspect-Oriented Software Development
  Provides a process and methodological approach for defining, specifying, designing, and constructing *aspects*
  Will be covered last week in «**Advanced Software Engineering**» topics.