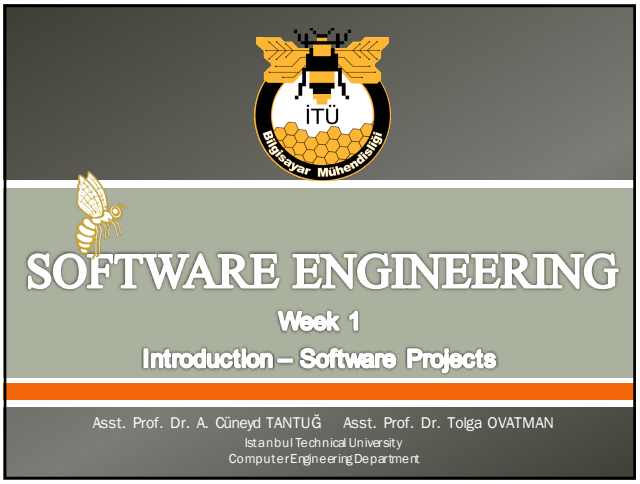
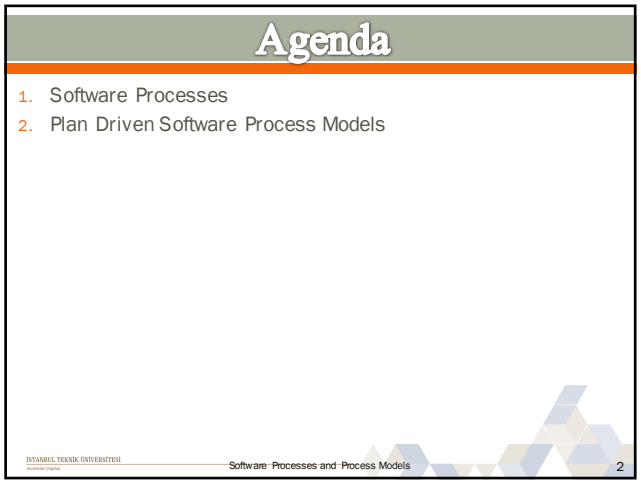
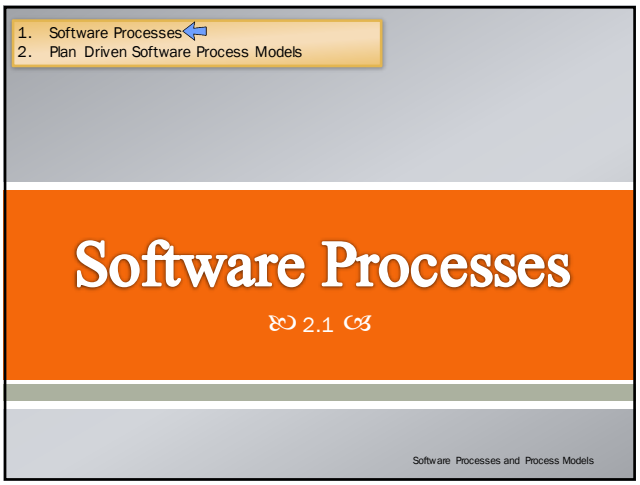
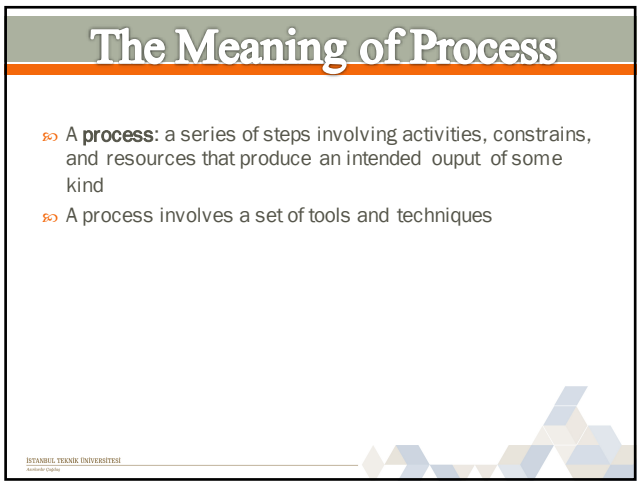

İTÜ
Bilgisayar Mühendisliği
SOFTWARE ENGINEERING
Week 1
Introduction – Software Projects
Asst. Prof. Dr. A. Cüneyd TANTUĞ Asst. Prof. Dr. Tolga OVATMAN
Istanbul Technical University
Computer Engineering Department


Agenda
1. Software Processes
2. Plan Driven Software Process Models
İSTANBUL TEKNİK ÜNİVERSİTESİ
Software Processes and Process Models
2


1. Software Processes
2. Plan Driven Software Process Models
Software Processes
2.1
Software Processes and Process Models


The Meaning of Process
A process: a series of steps involving activities, constrains, and resources that produce an intended ouput of some kind
A process involves a set of tools and techniques
İSTANBUL TEKNİK ÜNİVERSİTESİ

Reasons for Modeling a Process

- To form a common understanding
- To find inconsistencies, redundancies, omissions
- To find and evaluate appropriate activities for reaching process goals
- To tailor a general process for a particular situation in which it will be used

Software Life Cycle

- When a process involves building a software, the process may be referred to as software life cycle
  - Requirements analysis and definition
  - System (architecture) design
  - Program (detailed/procedural) design
  - Writing programs (coding/implementation)
  - Testing: unit, integration, system
  - System delivery (deployment)
  - Maintenance

Software Development

- Ideally, software is developed as described in
  - Linear
  - Starting from scratch
- In the real world, software development is totally different
  - We make mistakes
  - The client's requirements change while the software product is being developed

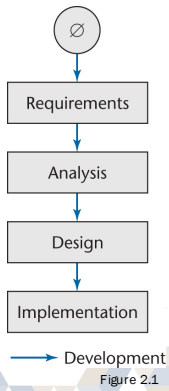


Figure 2.1

Moving Target Problem

- A change in the requirements while the software product is being developed
- Even if the reasons for the change are good, the software product can be adversely impacted
  - Dependencies will be induced
- Any change made to a software product can potentially cause a *regression fault*
  - A fault in an apparently unrelated part of the software
- If there are too many changes
  - The entire product may have to be redesigned and reimplemented
- Change is inevitable
  - Growing companies are always going to change
  - If the individual calling for changes has sufficient clout, nothing can be done about it
- There is no solution to the moving target problem

Iteration and Incrementation

- In real life, we cannot speak about "the analysis phase"
  - Instead, the operations of the analysis phase are spread out over the life cycle
- The basic software development process is iterative
  - Each successive version is intended to be closer to its target than its predecessor

Miller's Law

- At any one time, we can concentrate on only approximately seven *chunks* (units of information)
- To handle larger amounts of information, use *stepwise refinement*
  - Concentrate on the aspects that are currently the most important
  - Postpone aspects that are currently less critical
  - Every aspect is eventually handled, but in order of current importance
- This is an *incremental* process

Iteration and Incrementation (contd)

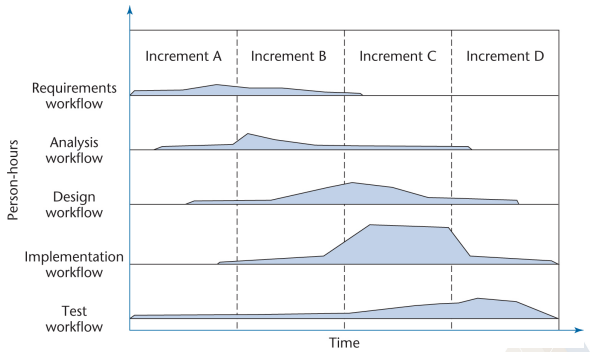


Figure 2.4

Iteration and Incrementation (contd)

- Iteration and incrementation are used in conjunction with one another
  - There is no single "requirements phase" or "design phase"
  - Instead, there are multiple instances of each phase

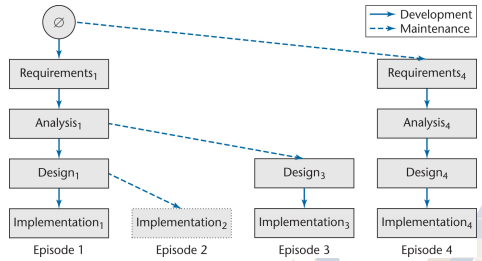


Figure 2.2 (again)

Classical Phases versus Workflows

- Sequential phases do not exist in the real world
- Instead, the five core workflows (activities) are performed over the entire life cycle
  - Requirements workflow
  - Analysis workflow
  - Design workflow
  - Implementation workflow
  - Test workflow

Workflows

- All five core workflows are performed over the entire life cycle
- However, at most times one workflow predominates
- Examples:
  - At the beginning of the life cycle
    - The requirements workflow predominates
  - At the end of the life cycle
    - The implementation and test workflows predominate
- Planning and documentation activities are performed throughout the life cycle

Iteration and Incrementation (contd)

- Iteration is performed during each incrementation

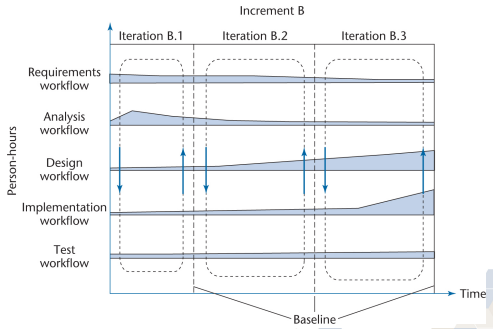


Figure 2.5

Iteration and Incrementation (contd)

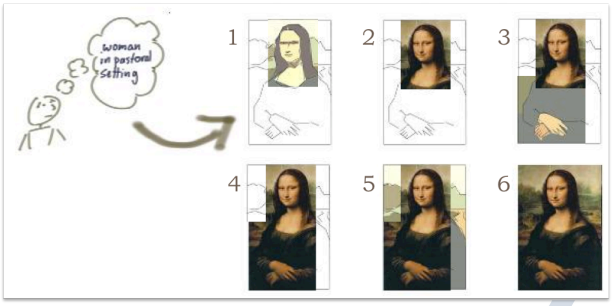


Figure 2.5

Risks and Other Aspects of Iter. and Increm.

- We can consider the project as a whole as a set of mini projects (increments)
- Each mini project extends the
  - Requirements artifacts
  - Analysis artifacts
  - Design artifacts
  - Implementation artifacts
  - Testing artifacts
- The final set of artifacts is the complete product

Risks and Other Aspects of Iter. and Increm. (contd)

- During each mini project we
  - Extend the artifacts (incrementation);
  - Check the artifacts (test workflow); and
  - If necessary, change the relevant artifacts (iteration)
- Each iteration can be viewed as a small but complete waterfall life-cycle model
- During each iteration we select a portion of the software product
- On that portion we perform the
  - Classical requirements phase
  - Classical analysis phase
  - Classical design phase
  - Classical implementation phase

Strengths of the Iterative-and-Incremental Model

- There are multiple opportunities for checking that the software product is correct
  - Every iteration incorporates the test workflow
  - Faults can be detected and corrected early
- The robustness of the architecture can be determined early in the life cycle
  - *Architecture* — the various component modules and how they fit together
  - *Robustness* — the property of being able to handle extensions and changes without falling apart

Strengths of the Iterative-and-Incremental Model (contd)

- We can *mitigate* (resolve) risks early
  - Risks are invariably involved in software development and maintenance
- We have a working version of the software product from the start
  - The client and users can experiment with this version to determine what changes are needed
- Variation: Deliver partial versions to smooth the introduction of the new product in the client organization
- The iterative-and-incremental life-cycle model is as regimented as the waterfall model ...
- ... because the iterative-and-incremental life-cycle model *is* the waterfall model, applied successively
- Each increment is a waterfall mini project

1. Software Processes
2. Plan Driven Software Process Models

Plan Driven Software Process Models

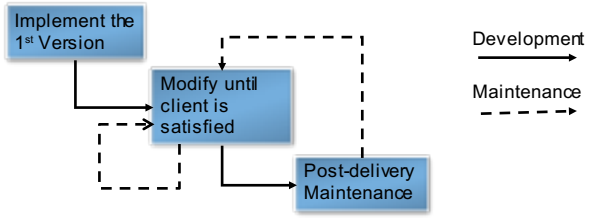
2.2

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

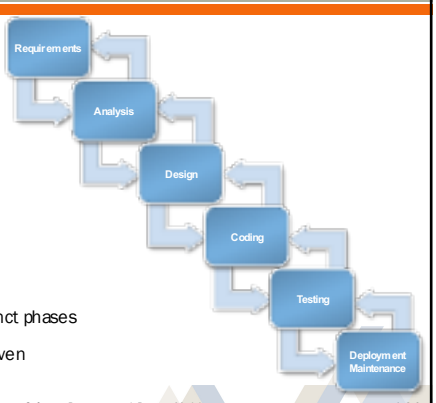
- In practice, most large systems are developed using a process that incorporates elements from all of these models.

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

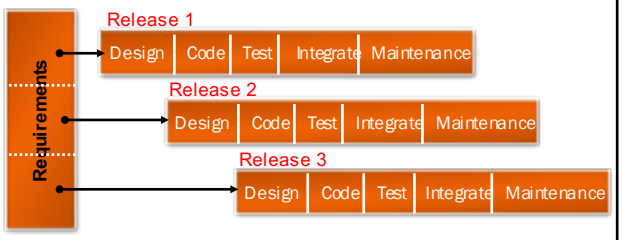
Pros

- 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.

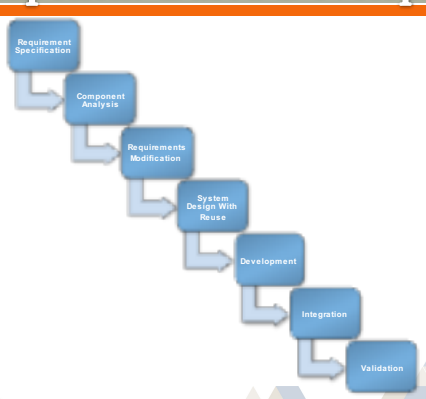
Cons

- 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

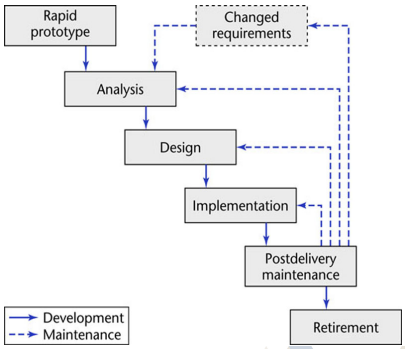
Pros

- 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.

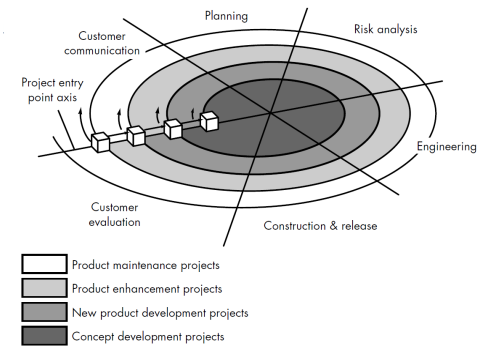
Cons

- 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.

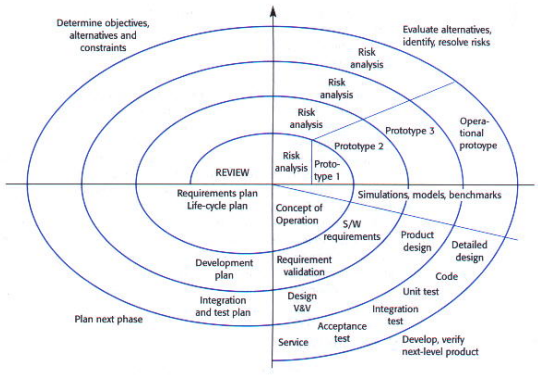
6. Spiral Model

- 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



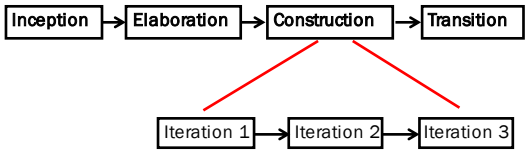
6. Spiral Model Sectors

- 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

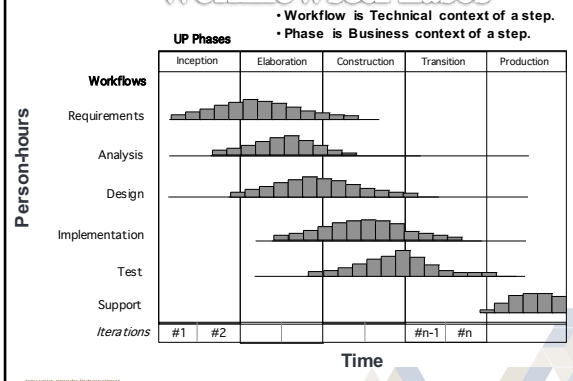
- The Unified Process (UP) is a "use-case driven, iterative and incremental" software process model closely aligned with Object-Oriented Analysis and Design.
- A modern generic process derived from the work on the UML and associated process.
- Brings together aspects of the 3 generic process models discussed 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.

Wrap-up

- Different life-cycle models have been presented
  - Each with its own strengths and weaknesses
- Criteria for deciding on a model include:
  - The organization
  - Its management
  - The skills of the employees
  - The nature of the product
- Best suggestion
  - "Mix-and-match" life-cycle model

Next Week

- We will discuss agile software development approach and practices in detail!