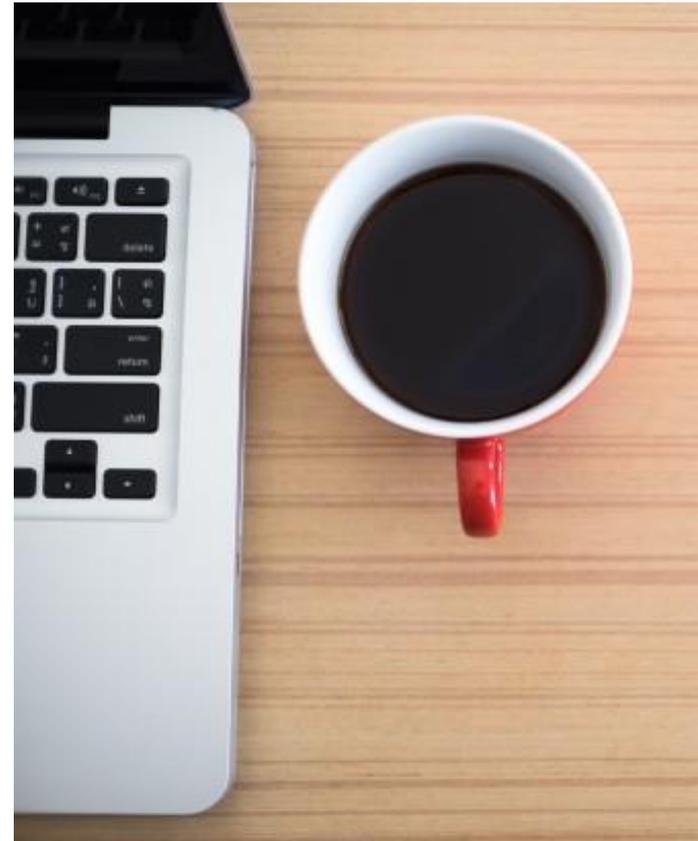


Course Topics

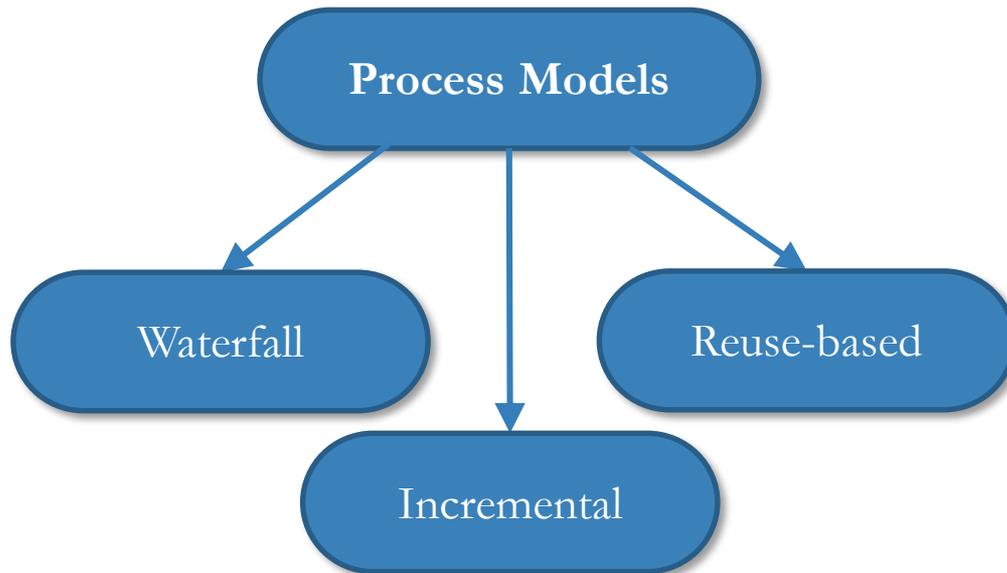
- Introduction
- Software Process Models
- Requirements Engineering
- Modeling
- Programming Languages
- Software Construction Techniques
- Testing
- Project Management
- Refactoring
- Ethical Issues

Lecture Objectives

- ✓ Coping with Change
 - Prototyping
 - Incremental delivery
 - Spiral Model
- ✓ Rational Unified Process
- ✓ Hybrid Process Models
- ✓ Agile Methods



Review



Process Activities

- 1- Specification
- 2- Development
- 3- Validation
- 4- Evolution

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.

Reducing the costs of rework

- **Change anticipation**, where 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**, where 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 to be altered to incorporate the change.
- Two ways to cope with change:
 - System prototyping
 - Incremental delivery

Prototyping



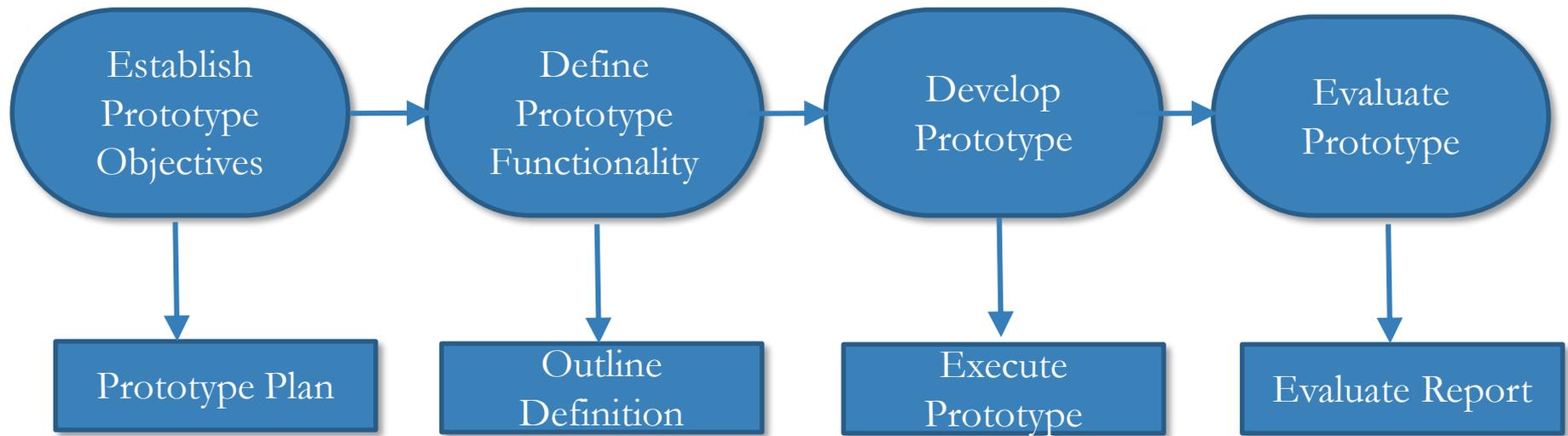
- A **prototype** is an initial version of a system used to demonstrate concepts and try out design options.
- A prototype can be used in:
 - The requirements engineering process to help with requirements elicitation and validation
 - In design processes to explore options and develop a UI design

Prototype development



- 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

The process of prototype development



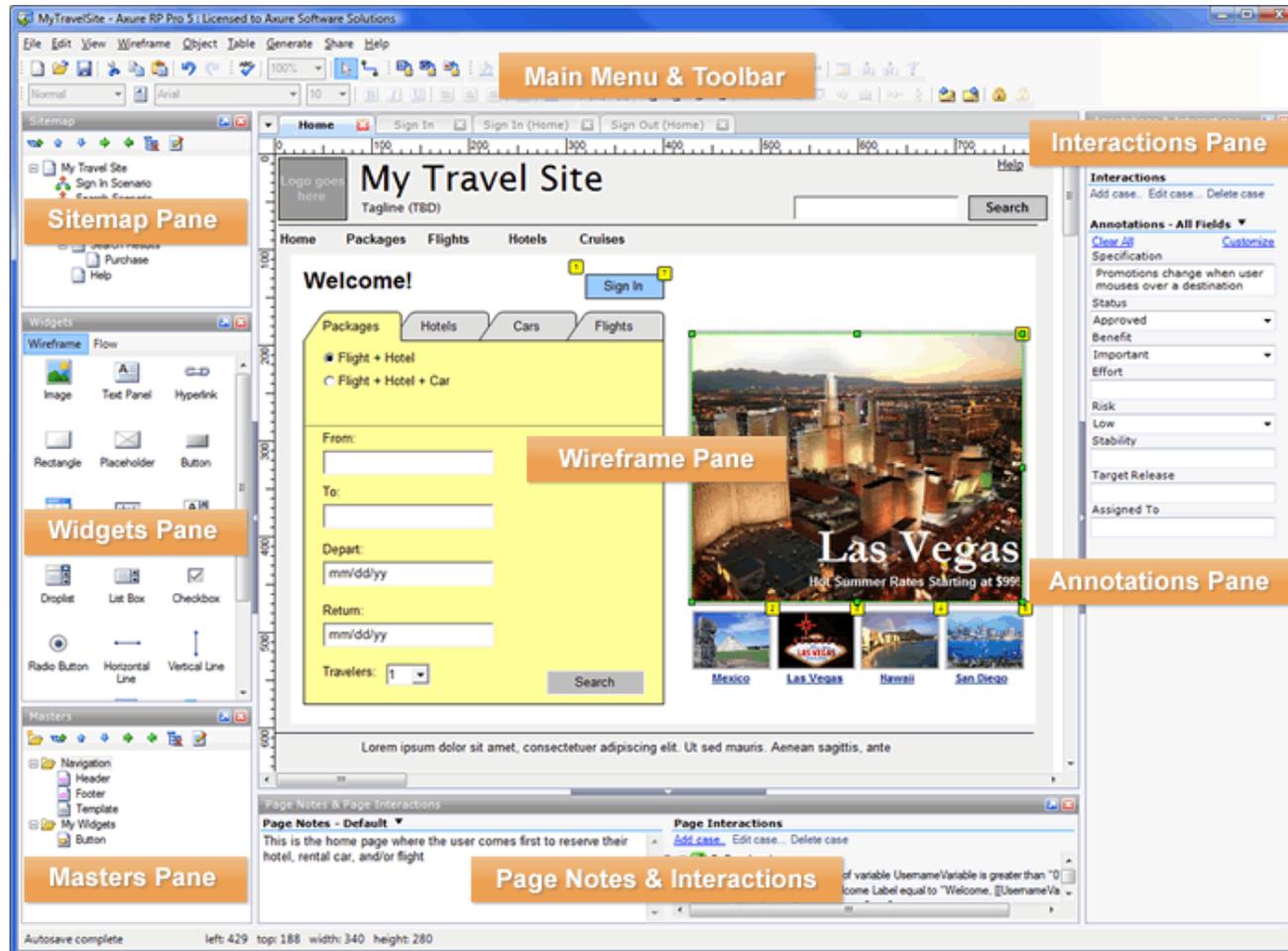
Benefits of Prototyping



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

Prototype development

- May be based on rapid prototyping languages or tools



Throw-away prototypes



Prototype + Extra stuff = Final System

Yes/ No? Why?



Throw-away prototypes



- Prototypes should be discarded after development as they are not a good basis for a production system:
 1. It may be impossible to tune the system to meet non-functional requirements
 2. Prototypes are normally undocumented
 3. The prototype structure is usually degraded through rapid change
 4. The prototype probably will not meet normal organizational quality standards

Prototyping



■ Problems

- It may be impossible to tune the prototype to meet non-functional requirements,
- Changes degrades the system structure.
- Prototype is undocumented.
- Special skills (e.g. in languages for rapid prototyping) may be required

■ Applicability

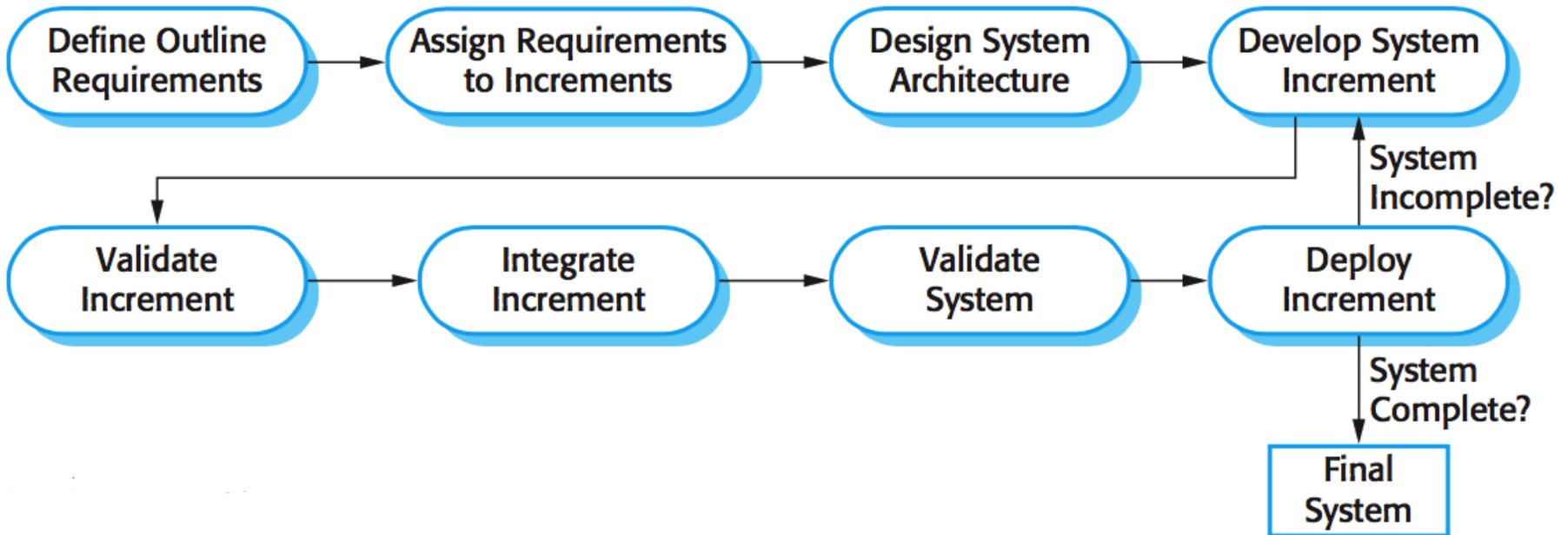
- For small or medium-size interactive systems
 - For parts of large systems (e.g. the user interface)
 - For short-lifetime systems
- 

Incremental 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 **prioritised** and the highest priority requirements are included in early increments.
 - Once the development of an increment is started, the requirements for the services to be delivered in the first increment are defined in detail. Further requirements for later increments can continue to evolve.
- 

Incremental Delivery



Incremental Delivery



■ Advantages

- Customers can use the early increments as prototypes and gain experience that informs their requirements for later system increments.
- Customers do not have to wait until the entire system is delivered
- Easy to incorporate changes into the system.
- As the highest-priority services are delivered first and increments then integrated, the most important system services receive the most testing

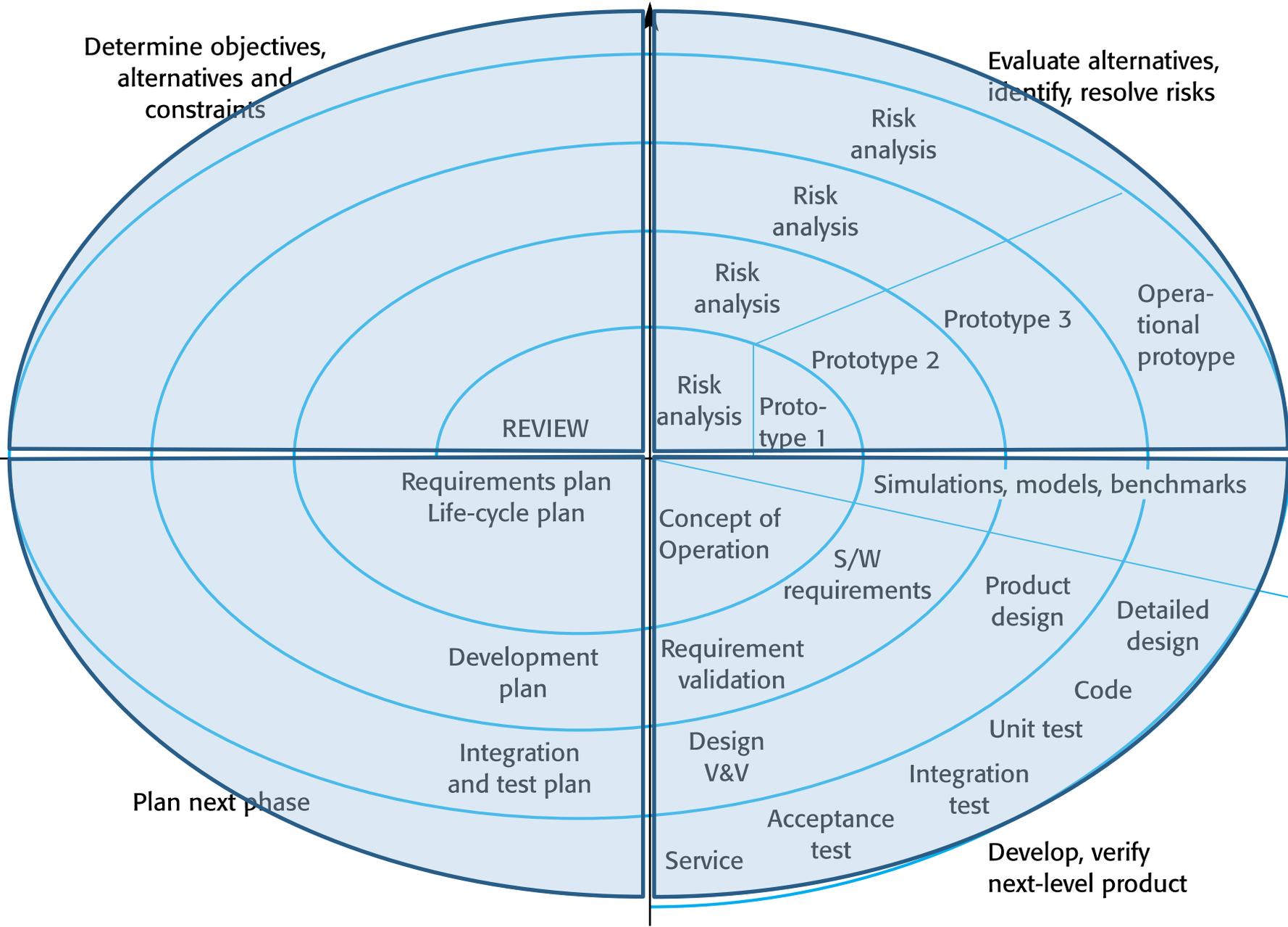
■ Issues

- Iterative development can also be difficult when a replacement system is being developed.
 - Conflicts with the procurement model of many organizations, where the complete system specification is part of the system development contract.
- 

Boehm's spiral model



- Risk-driven development process
 - proposed by Boehm in 1988
 - 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.
- 



Spiral model – Typical Traversal



- Identify the objectives, alternatives or constraints for each cycle of the spiral.
 - Evaluate the alternative relative to the objectives and constraints.
 - In this step, many of the risks are identified and evaluated.
 - Depending on the amount of and type of identified risks,
 - Develop a prototype, more detailed evaluation,
 - An evolutionary development, or some other step to further reduce the risk of achieving the identified objective.
 - On the other hand, if risk is substantially reduced, the next step may just be a task such as requirements, design or code.
 - Validate the achievement of the objective and plan for next cycle.
- 

Spiral model usage



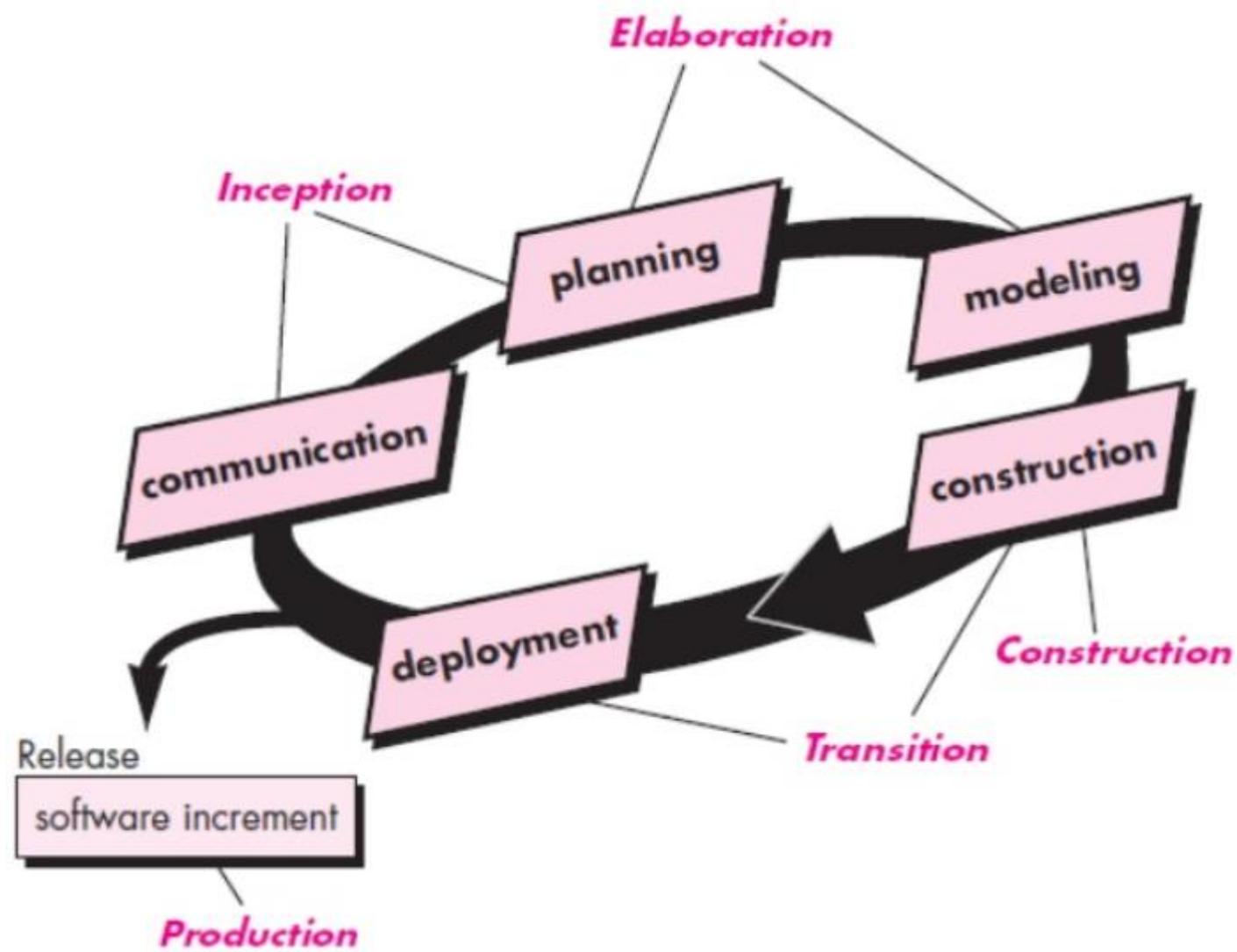
- Spiral model has been very influential in helping people think about iteration in software processes and introducing the risk-driven approach to development.
- In practice, however, the model is **rarely** used as published for practical software development.

The Rational Unified Process



- Modern process model
 - Closely aligned with the Unified Modeling Language (UML).
- “Use-case” driven
- Architecture-centric
- Iterative and incremental software process
- Intended for large-scale applications where robustness, scalability, and extensibility are mandatory
 - Telecommunication, financial services, transportation, etc.

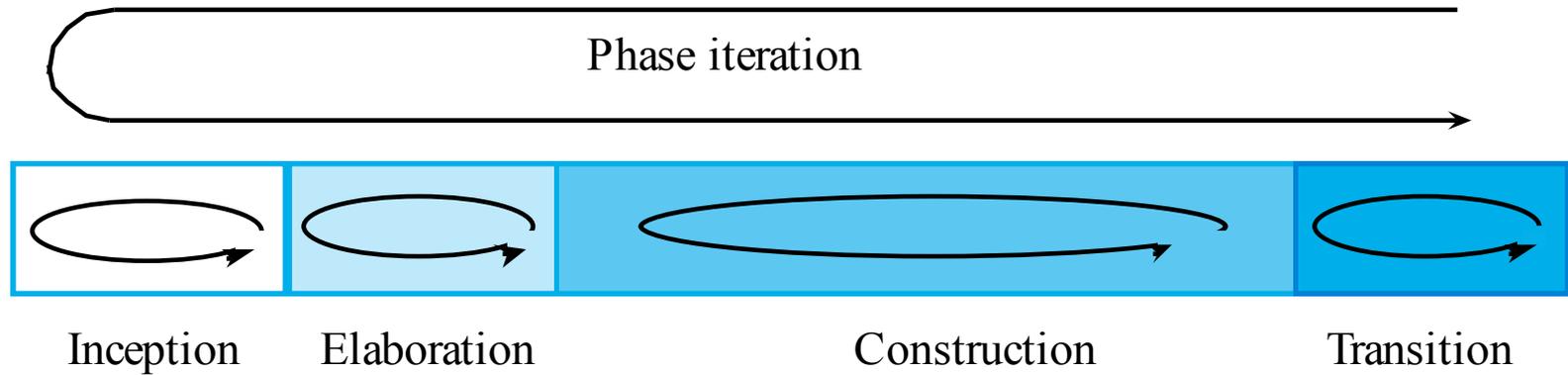
The Rational Unified Process



RUP—Dynamic Perspective



- Phase Model

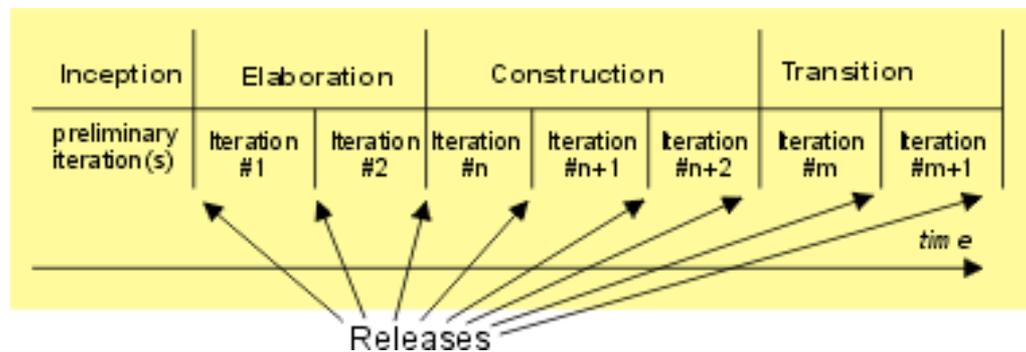
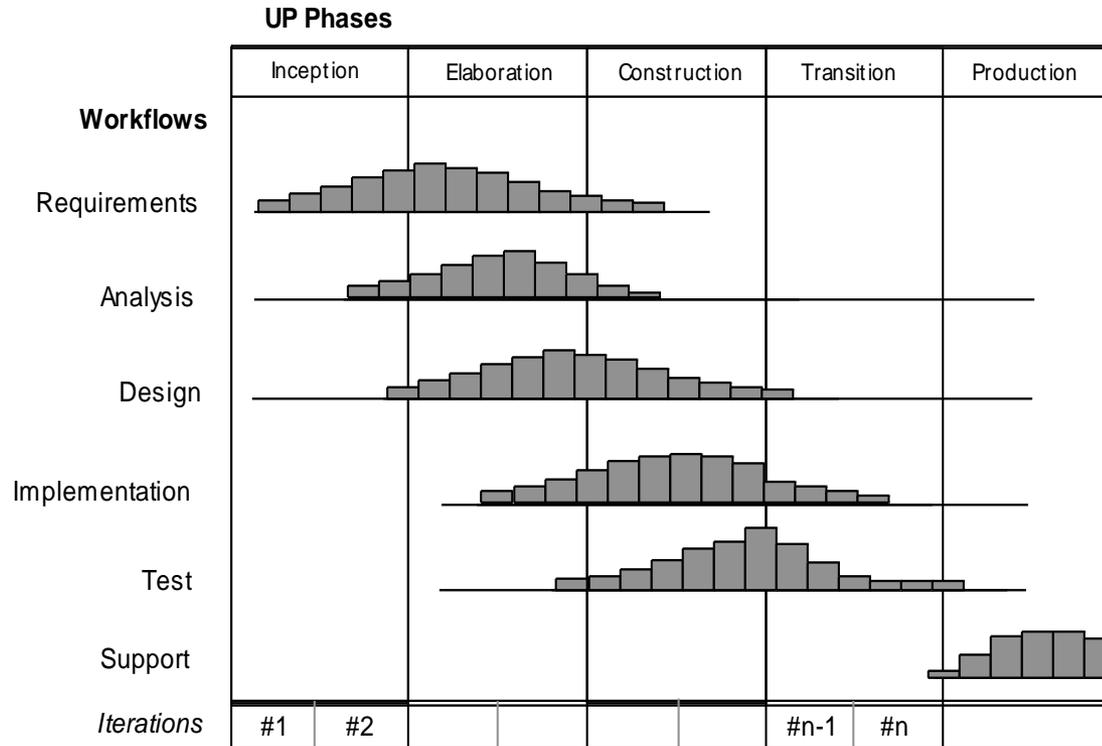


RUP Phases



- Inception
 - Establish the business case for the system {Identifies the External Entities that interact with the system}.
- Elaboration
 - Develop an understanding of the problem domain and the system architecture {Results in requirements model, architectural description, and a development plan}.
- Construction
 - System design, programming, and testing {Results in a working software system and associated documentation that ready for delivery to the user}.
- Transition
 - Deploy the system in its operating environment {Results in a documented software system that is working correctly in its operational environment}.

RUP Phases



RUP good practice



- Develop software iteratively
- Manage requirements
- Use component-based architectures
- Visually model software
- Verify software quality
- Control changes to software

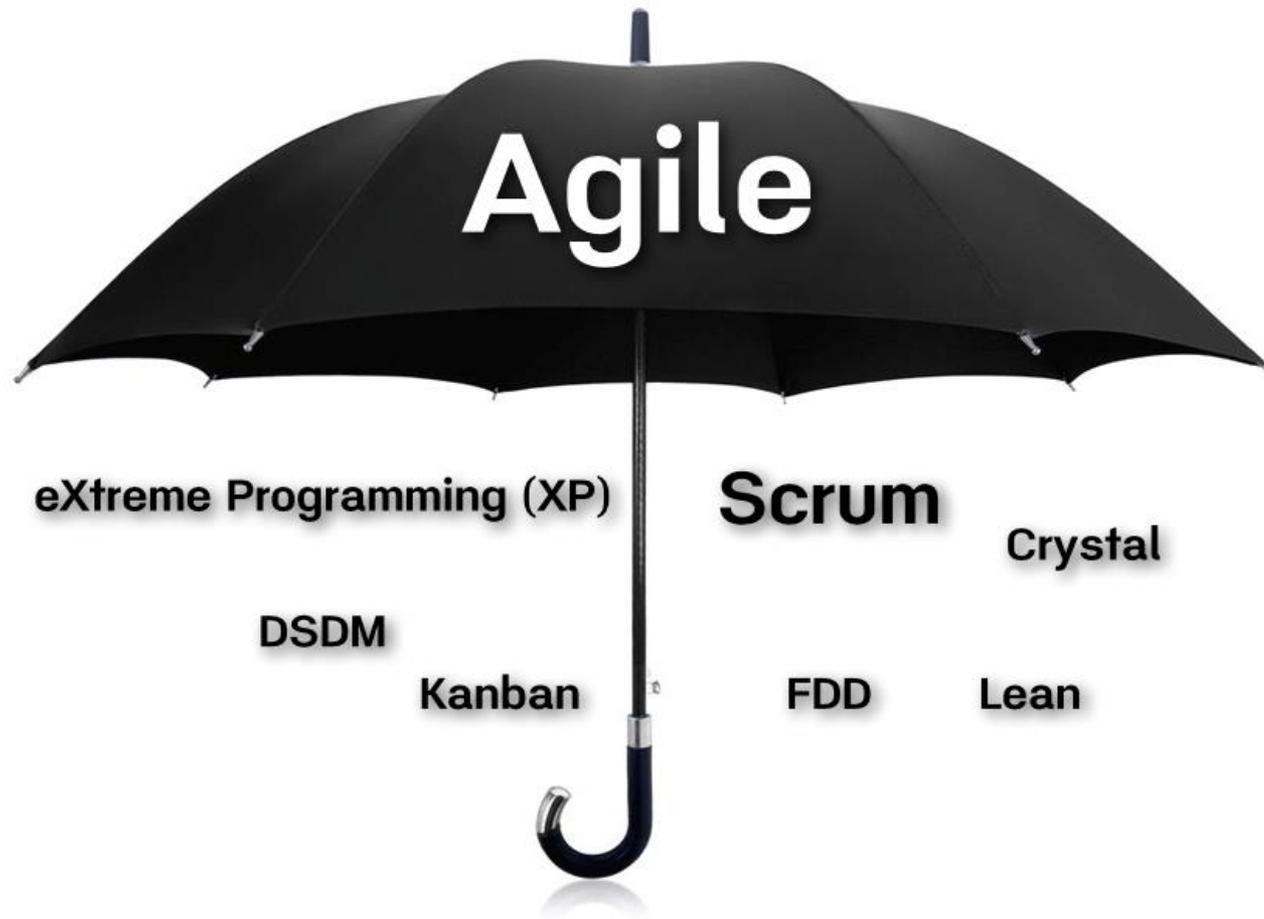
Hybrid Process Models



- Large systems are usually made up of several sub-systems

- The same process model need not be used for all subsystems
 - Prototyping for high-risk specifications
 - Waterfall model for well-understood developments

Agile Methods



Agile methods: Extreme Programming (XP)



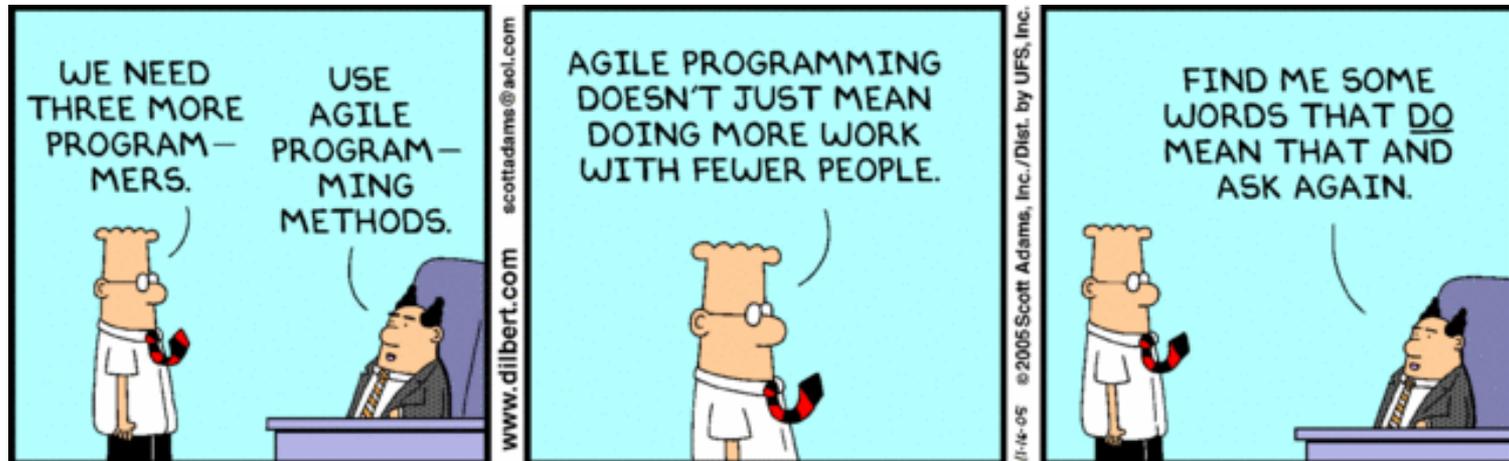
- Three to ten **programmers** working at one location.
 - One or more **customers** are on site.
 - Development occurs in frequent builds or iterations, each of which is releasable and delivers incremental user functionality.
 - The unit of requirements gathering is the **user story**, a chunk of functionality that provides value to the user. User stories are written by the customers on site.
- 

Agile Methods: XP Principles Characteristics

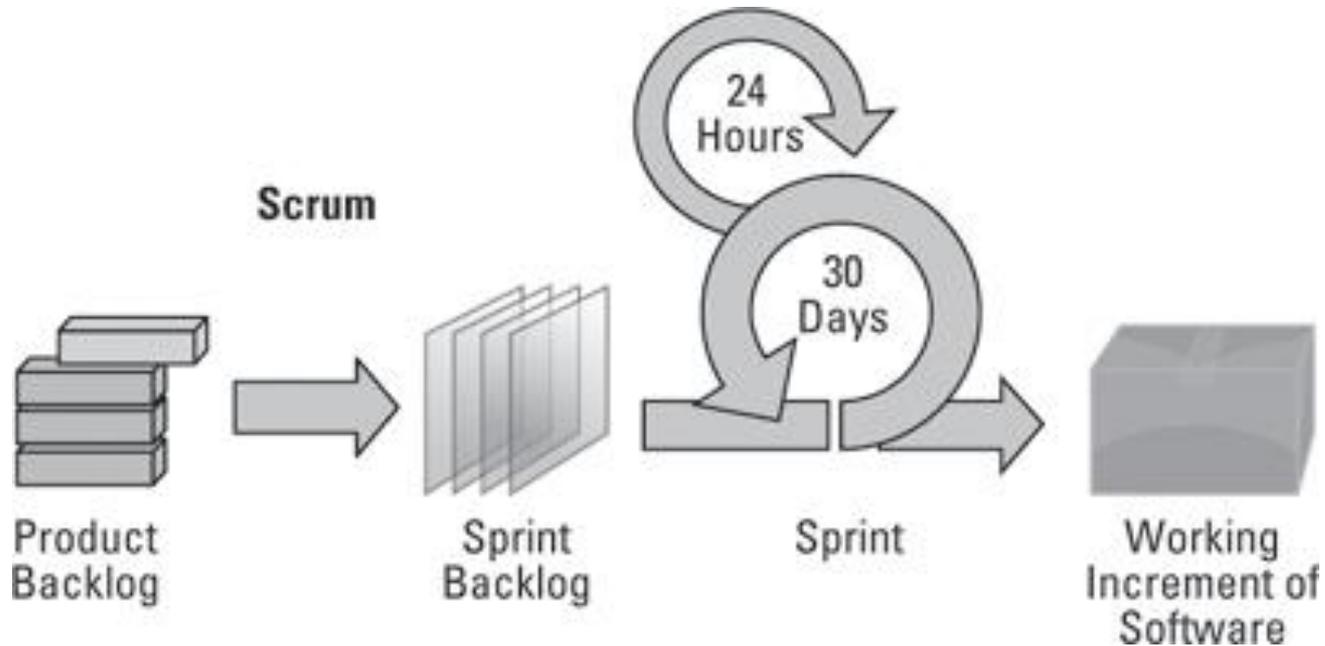


- Programmers work in pairs and follow strict coding standards. They do their own unit **testing** and are supposed to routinely **refactor** the code to keep the design simple.
 - Since little attempt is made to understand or document future requirements, the code is constantly refactored (redesigned) to address changing user needs.
- 

Agile Methods: Extreme Programming (XP)



Agile Methods: Scrum



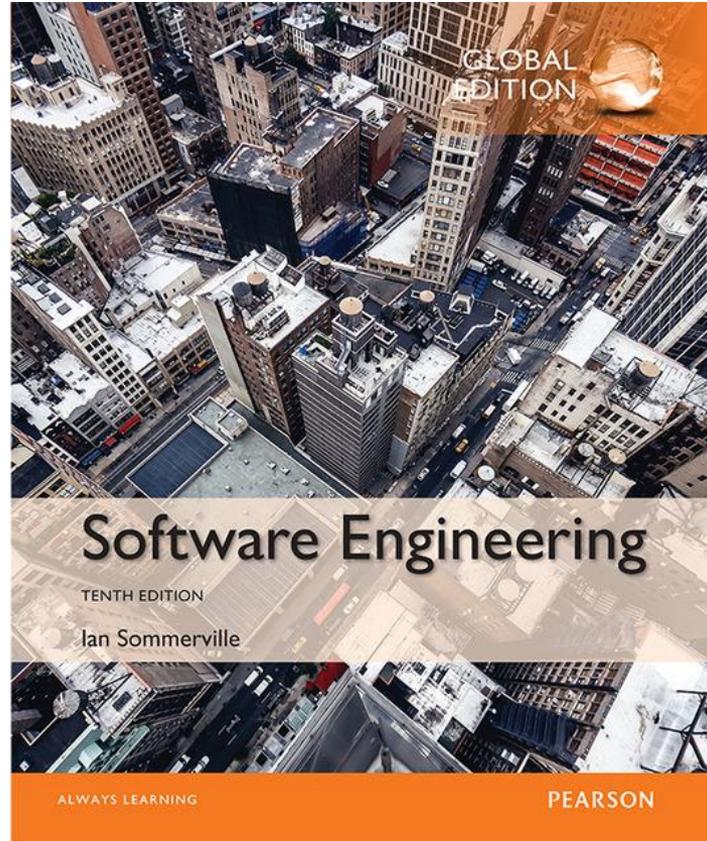
- Work is done in “sprints,” which are timeboxed iterations of a fixed 30 days or fewer duration.
- Work within a sprint is **fixed**. Once the scope of a sprint is committed, no additional functionality can be added, except by the development team.

Key points



- Processes should include activities to cope with change. This may involve a prototyping phase that helps avoid poor decisions on requirements and design.
- Processes may be structured for iterative development and delivery so that changes may be made without disrupting the system as a whole.

Read



Chapter 2 and Chapter 3

References



- Ian Sommerville, “Software Engineering”, 10th Edition, Addison-Wesley, 2015.
- Timothy C. Lethbridge and Robert Laganière, “Object-Oriented Software Engineering: Practical Software Development using UML and Java”, 2nd Edition, McGraw Hill, 2001.
- R. S. Pressman, Software Engineering: A Practitioner’s Approach, 10th Edition, McGraw-Hill, 2005.

Next



Chapter 4 Requirements Engineering



Course Topics

- ~~Introduction~~
- ~~Software Process Models~~
- Requirements Engineering
- Modeling
- Programming Languages
- Software Construction Techniques
- Testing
- Project Management
- Refactoring
- Ethical Issues