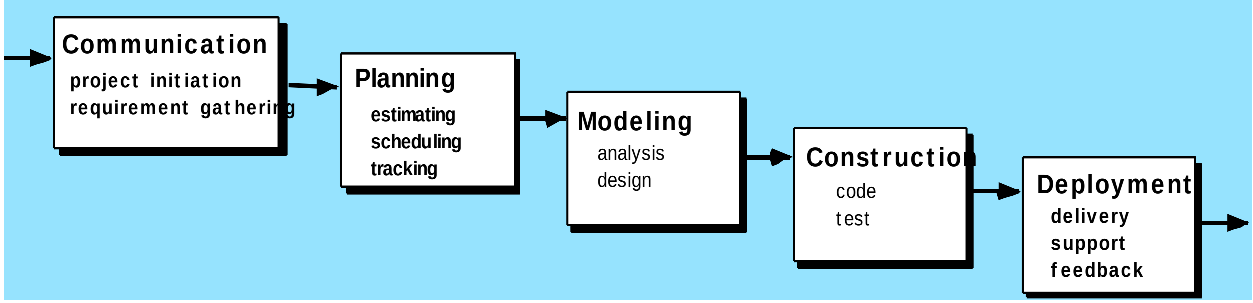
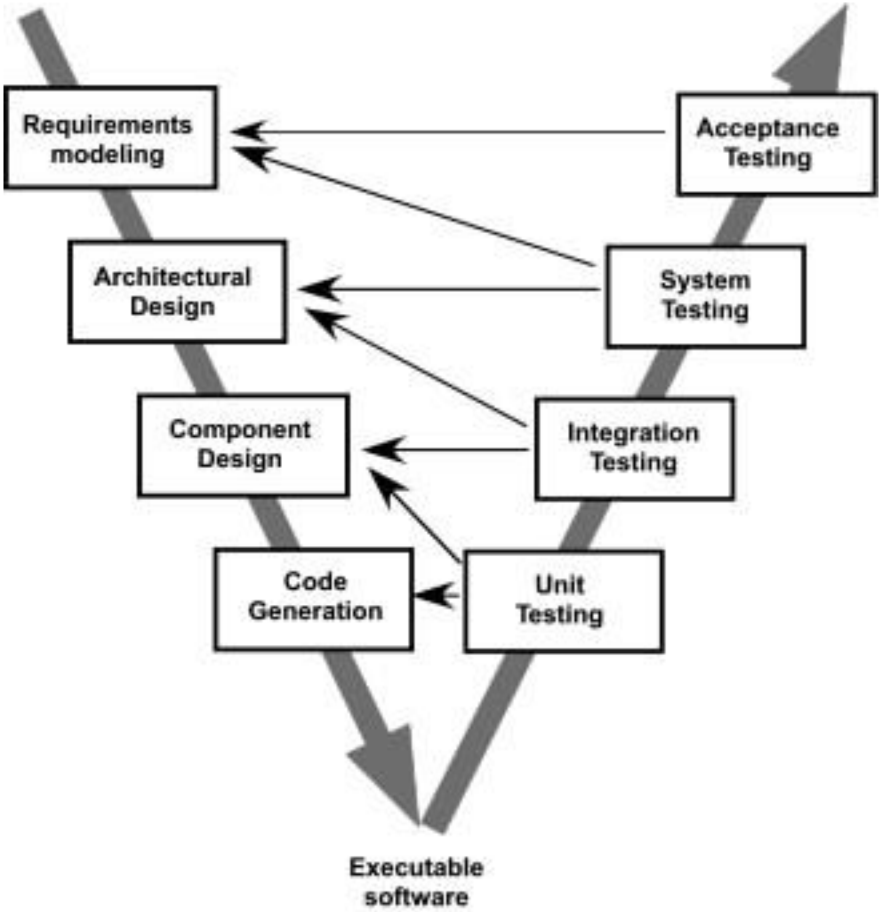
IT 242 FOR Final Exam 2016-2017 by: Ghannam

Chapter 4&5

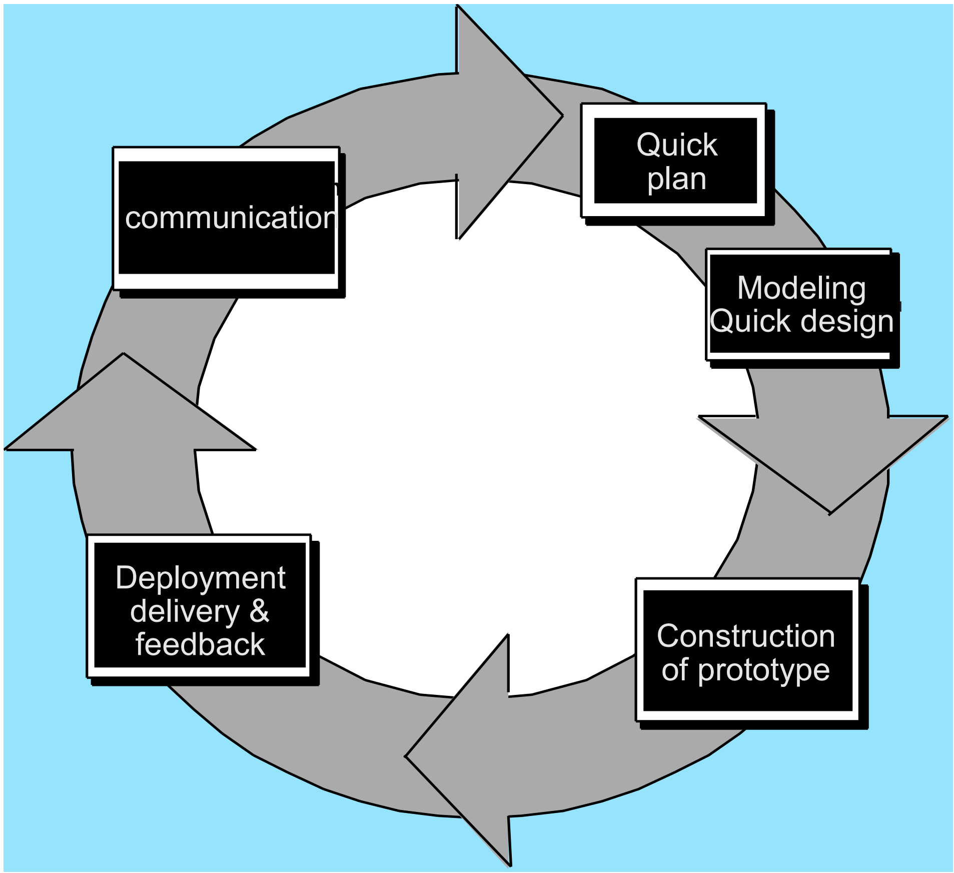
The Waterfall Model



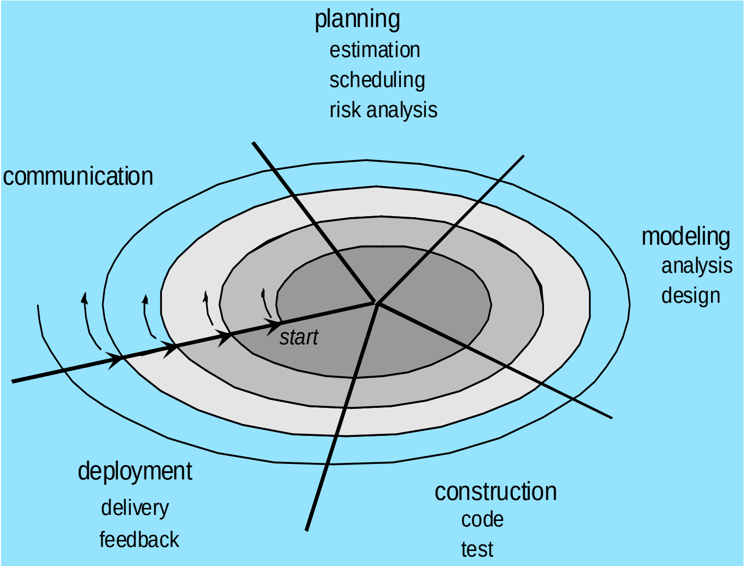
The V-Model



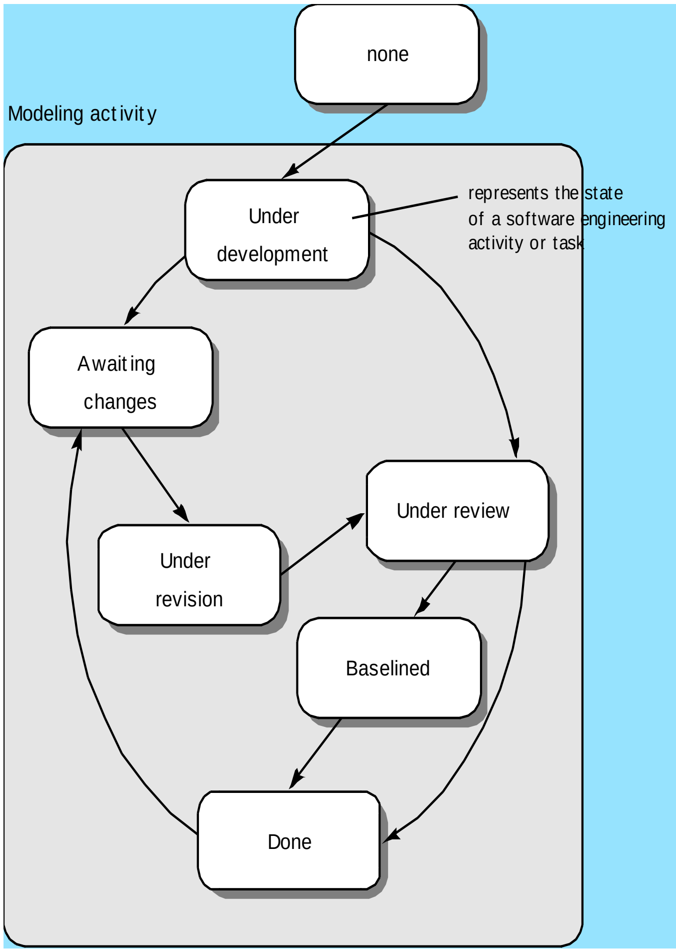
Evolutionary Models: Prototyping



Evolutionary Models: The Spiral



Evolutionary Models: Concurrent



Extreme Programming (XP): The most widely used agile process, originally proposed by Kent Beck

XP Rules:

- **XP Planning**
  - **Create user stories**
  - **Assigns cost to each story**
  - **Release planning creates the release schedule.**
  - **A commitment is made on delivery date**
  - **The project is divided into iterations.**
- **XP Designing**
  - **Simplicity**
  - **Use CRC cards**
  - **Create spike solutions to reduce risk.**
  - **Refactor whenever and wherever possible.**
- **XP Coding**
  - **Code the unit test first.**
  - **pair programming**
- **XP Testing**
  - **All unit tests are executed daily**
  - **Acceptance tests are run often and the score is published.**

Chapter 11

- **Passive state is simply the current status of all of an object's attributes.**
- **Active state of an object indicates the current status of the object as it undergoes a continuing transformation or processing.**
- **Content objects are extracted from use-cases**
- **Attributes of each content object are identified**
- **Relationships among content objects and/or the hierarchy of content maintained by a WebApp**
- **Relationships—entity-relationship diagram or UML**
- **Hierarchy—data tree or UML**

Interaction Model:

1. **use-cases**
2. **sequence diagrams**
3. **state diagrams**
4. **a user interface prototype**

Chapter 12

Software Engineering Design:

1. **Data/Class design**: transforms analysis classes into implementation classes and data structures
2. **Architectural design**: defines relationships among the major software structural elements
3. **Interface design**: defines how software elements, hardware elements, and end-users communicate
4. **Component-level design**: transforms structural elements into procedural descriptions of software components

Fundamental Concepts:

- **Abstraction:** data, procedure, control
- **Architecture:** the overall structure of the software
- **Patterns:** “conveys the essence” of a proven design solution
- **Separation of concerns:** any complex problem can be more easily handled if it is subdivided into pieces
- **Modularity:** compartmentalization of data and function
- **Hiding:** controlled interfaces
- **Functional independence:** single-minded function and low coupling
- **Refinement:** elaboration of detail for all abstractions
- **Aspects:** a mechanism for understanding how global requirements affect design
- **Refactoring:** a reorganization technique that simplifies the design
- **OO design concepts**
- **Design Classes:** provide design detail that will enable analysis classes to be implemented

Separation of Concerns: Any complex problem can be more easily handled if it is subdivided into pieces that can each be solved and/or optimized independently, a problem takes less effort and time to solve.

Concern is a feature or behavior that is specified as part of the requirements model

Modularity is the single attribute of software that allows a program to be intellectually manageable

Functional Independence: achieved by developing modules with "single-minded" function and an "aversion" to excessive interaction with other modules

Cohesion is an indication of the relative functional strength of a module

Coupling is an indication of the relative interdependence among modules.

Design classes:

- Entity classes
- Boundary classes
- Controller classes

Inheritance: all responsibilities of a superclass is immediately inherited by all subclasses
Messages: stimulate some behavior to occur in the receiving object
Polymorphism: a characteristic that greatly reduces the effort required to extend the design

Chapter 13

Why Architecture?

- analyze the effectiveness of the design
- consider architectural alternatives
- Reduce the risks

Architectural Styles

- a set of components
- a set of connectors
- constraints
- semantic models

Architectural parts:

- Data-centered architectures
- Data flow architectures
- Call and return architectures
- Object-oriented architectures
- Layered architectures

Archetype: is an abstraction (similar to a class) that represents one element of system behavior

Architectural description language (ADL): provides a semantics and syntax for describing a software architecture. With the ability to:

- decompose architectural components
- compose individual components into larger architectural blocks
- Represent interfaces between components.

Chapter 22:

Verification refers to the set of tasks that ensure that software correctly implements a specific function. (Are we building the product right?)
Validation refers to a different set of tasks that ensure that the software that has been built is traceable to customer requirements (Are we building the right product?)

General Testing Criteria:

- **Interface integrity:** internal and external module interfaces are tested as each module or cluster is added to the software
- **Functional validity:** test to uncover functional defects in the software
- **Information content:** test for errors in local or global data structures
- **Performance**: verify specified performance bounds are tested

OO Testing Strategy:

- class testing is the equivalent of unit testing
  - operations within the class are tested
  - the state behavior of the class is examined
- integration applied three different strategies
  - thread-based testing
  - use-based testing
  - cluster testing

MobileApp Testing:

- User experience testing
- Device compatibility testing
- Performance testing
- Connectivity testing
- Security testing
- Testing-in-the-wild
- Certification testing

High Order Testing:

- Validation testing
- System testing
- Alpha/Beta testing
- Recovery testing
- Security testing
- Stress testing
- Performance Testing

Chapter 25

Testing Quality Dimensions:

- **Content**: is evaluated at both a syntactic and semantic level
- **Function**: is tested for correctness, instability, and general conformance to appropriate implementation standards
- **Structure**: is assessed to ensure that it
  - properly delivers WebApp content and function
  - is extensible
  - can be supported as new content or functionality is added
- **Usability**: is tested to ensure that each category of user
- **Navigability**: is tested to ensure that all navigation syntax and semantics are exercised to uncover any navigation errors
- **Performance**: is tested under a variety of operating conditions, configurations, and loading to ensure that
  - the system is responsive to user interaction
  - the system handles extreme loading without unacceptable operational degradation
- **Compatibility**: is tested by executing the WebApp in a variety of different host configurations on both the client and server sides.
- **Interoperability**: is tested to ensure that the WebApp properly interfaces with other applications and/or databases.
- **Security**: is tested by assessing potential vulnerabilities and attempting to exploit each.

WebApp Testing Strategy:

- The content model for the WebApp is reviewed to uncover errors.
- The interface model is reviewed to ensure that all use-cases can be accommodated.
- The design model for the WebApp is reviewed to uncover navigation errors.
- The user interface is tested to uncover errors in presentation and/or navigation mechanics.
- Selected functional components are unit tested.

Testing Interface Mechanisms:

- common gateway interface (CGI scripts)
- Streaming content
- Cookies (block of data sent by the server and stored by a browser as a consequence of a specific user interaction)
- Application specific interface mechanisms (macro)

Compatibility Testing:

- Compatibility testing: is to define a set of ―commonly encountered‖ client side computing configurations and their variants
- Create a tree structure identifying
  - each computing platform
  - typical display devices
  - the operating systems supported on the platform
  - the browsers available
  - likely Internet connection speeds
  - Similar information.
- Derive a series of compatibility validation tests

Chapter 36

Maintainable Software:

The design and implementation of the software must "assist" the person who is making the change

- Maintainable software exhibits effective modularity
- It makes use of design patterns that allow ease of understanding
- It has been created by software engineers who recognize that they may not be around when changes must be made.

Software Supportability:

- The capability of supporting a software system over its whole product life
- The software should contain facilities to assist support personnel when a defect is encountered in the operational environment
- Support personnel should have access to a database that contains records of all defects that have already been encountered

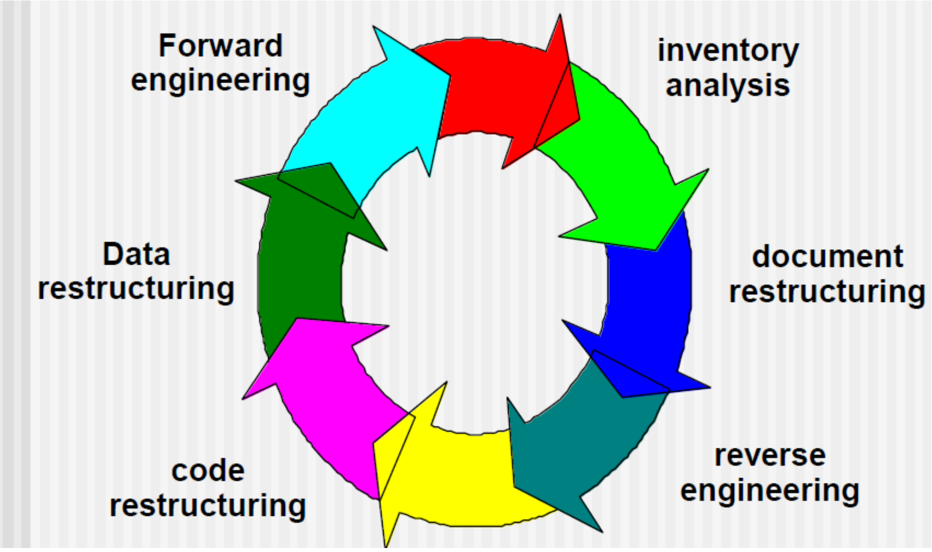
Business Process Reengineering:

- Business definition
  - cost reduction
  - time reduction
  - quality improvement
  - Personnel development and empowerment.
- Process identification
- Process evaluation
- Process specification and design
- Prototyping
- Refinement and instantiation

Business process reengineering (BPR Principles):

- Organize around outcomes, not tasks.
- Have those who use the output of the process perform the process.
- Link parallel activities instead of integrated their results.
- Put the decision point where the work is performed, and build control into the process.
- Capture data once, at its source.

Software Reengineering

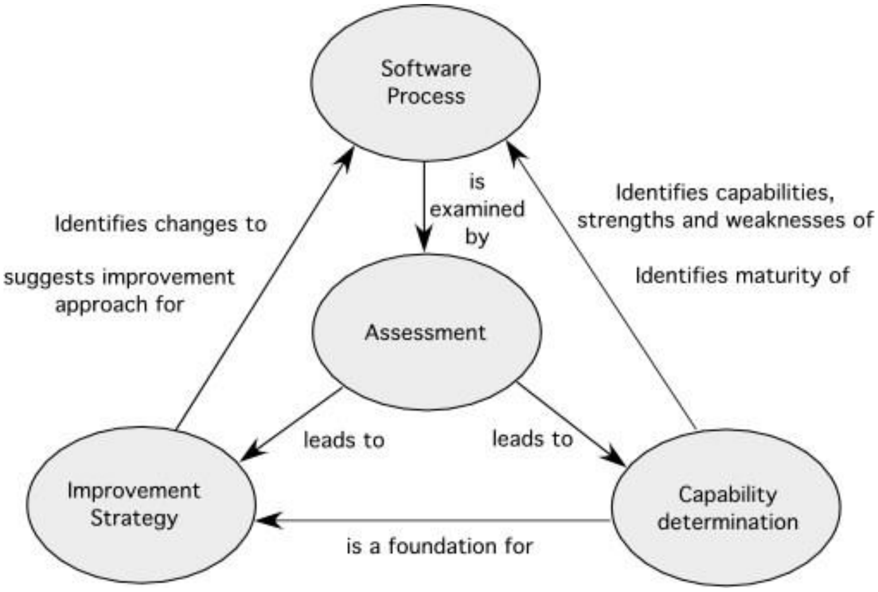


Code Restructuring:

- Source code is analyzed using a restructuring tool.
- Poorly design code segments are redesigned
- Violations of structured programming constructs are noted and code is then restructured
- The resultant restructured code is reviewed and tested to ensure that no anomalies have been introduced
- Internal code documentation is updated

Chapter 37

Elements of a SPI Framework:



Maturity Models:

- Is applied within the context of an SPI framework.
- The intent of the maturity model is to provide an overall indication of the "process maturity" exhibited by a software organization

The SPI Process:

- **Installation/Migration**
  - **Software process redesign (SPR):** is concerned with identification, application, and refinement of new ways to dramatically improve and transform software processes.
  - The existing ("as-is") process.
  - A transitional ("here-to-there") process.
  - The target ("to be") process.
- **Evaluation**
  - Assesses the degree to which changes have been instantiated and adopted.
  - The degree to which such changes result in better software quality or other tangible process benefits.
  - The overall status of the process and the organizational culture as SPI activities proceed.
- From a **qualitative point of view**, past management and practitioner attitudes about the software process can be compared to attitudes polled after installation of process changes.