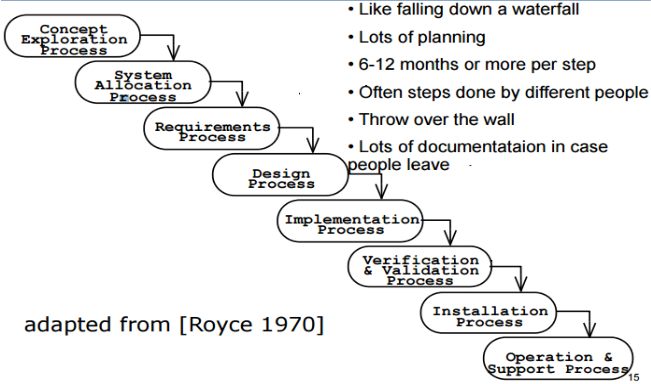
What is Software Engineering?

* Software Engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification though to maintaining the system after it has gone into use.
* IEEE Standard 610.12 – The application of a systematic, discipline, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

Proposal Key Points

* Invent, Emphasize more on content rather than presentation, Focus on customer needs and wants, The key is to diagnose the problem, Customer benefits, Customer impacts

Three Key Components

* Diagnose the problem, Prescribe the Treatment, Describe Plan of Work

Four Primary Software Engineering Activities

* Specifications, Development, Validation, Evolution

Waterfall Model

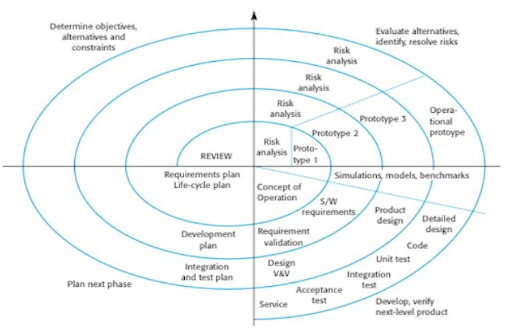
* One of the first published models, Separate and distinct phases are performed in sequence phases are performed in sequence, Planning occurs up front: “Plan driven”

The separate phases

* Requirements definition, Software design, Implementation, Testing, Maintenance

The output of one stage is input to the next

Tends to require/generate much documentation.

* Concept Exploration Process, System Allocation Process, Requirements Process, Design Process, Implementation Process, Verification Process, Installation Process, Operation and Support process
* Like falling down a waterfall, lots of planning, 6-12 months or more per step, Often steps done by different people, Throw over the wall, Lots of documentation in case people leave.

Spiral Model

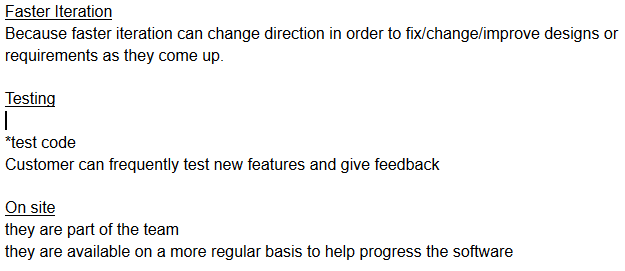
* Proposed by Barry Boehm in 1988
* Process represented as a spiral
  + Each loop represents a phase in the process
  + Content of each phase is not predetermined, plan as you go.
* Risks are explicitly assessed and resolved.
  + Assumes need for change are a result of project risks
* Sectors of the model:
  + Identify objectives, alternatives and constraints
    - Evaluate and reduce risks (May develop prototype)
    - Development and Validation
    - Plan next phase (after review of current phase)

Spiral Model Issues

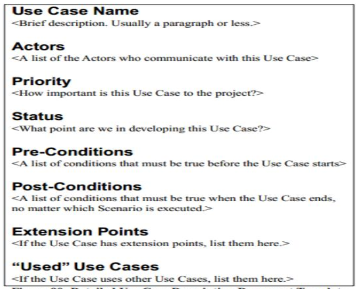
* Good for high risks projects
  + Often used in combination with other process models
* In practice, the model is rarely used as published
* Somewhat similar to incremental development, but
  + Risk assessment is incorporated into the process
  + Development is not required to be incremental
    - Prototypes and results of previous loops can be discarded.
    - Production development could be postponed until the last loop.

Agile Manifesto: Individuals and Interactions, Working software, Customer collaboration, Responding to change

Agile Process Pros

* Efficient Handling of changes in requirements
* Low process complexity
* Low cost overhead
* Fast Results (Short Iterations, Core System produced upfront)
* Useable Systems that are acceptable due to customer involvement
* From incremental development
  + very small increments (2 – 3 weeks)
  + customers evaluate versions
* Minimal process documentation
  + Minimal user requirements documents
  + Lack of detailed design specifications
* Focus on human and team aspects of software development
* Favor use of development tools like IDE’s

Agile Process Cons

* Difficult to scale methods to large systems
* Heavy on teamwork
* Reliance on frequent access to customer
* Cultural clash: Nonconventional process and management
* Not suited for security or critical systems

XP

* Onsite customer
* The Planning Game
* Small releases
* Testing
* Simple design
* Refactoring
* Metaphor
* Pair programming
* Collective ownership
* Continuous integration
* 40 hour week
* Coding standards

Use Case Diagram

* Use case name – Brief description, usually a paragraph or less.
* Actors – A list of the actors who communicate with this use case.
* Priority – How important is the use case to the project.
* Status – What point are we in developing this use case.
* Pre-conditions – A list of conditions that must be true before the use case statements.
* Post-conditions – A list of conditions that must be true when the use case ends, no matter which scenario is executed.

Incremental Development

* Several development iterations are performed in sequence
* Each iteration is a self-contained mini-project composed of activities such as requirements analysis, design, programming, and test.
* Each iteration produces a new version (called increment)
  + Each version adds functionality to the previous version.
  + Only the final version is a complete version is a complete system.
* Each version is exposed to the user for feedback.
  + The customer may come to the developers site for demos/testing.
  + If the intermediate versions are given to the customer, it is called incremental Delivery.
* Reduces cost of accommodating changing customer requirements.
  + Early versions are incomplete, so less re-work to do.
  + May require no changes to current version (add to future version)
* It is easier to get customer feedback.
  + Users understand a working incremental release better than documents from the specification or design phase.
* Does not need to be planned entirely up front.
* Early versions can be implemented the most important urgent, or risky features.
* The process is not visible
  + There’s less process documentation, so it’s difficult to measure progress.
  + May not know how many more increments are required.
* Difficult to design and implement common faculties need by all versions.
* System structure tends to degrade as new increments are added.
  + Makes code more difficult to modify.
  + Refactoring – disciplined techniques for restricting an existing body of code, altering its internal structure without changing its external behavior.
  + Modifying a program to improve its structure, reduce its complexity, or make it easier to understand.

Types of Software components for reuse

* Web Services
  + Various “functions” available for remote invocation from apps
  + Example: Weather API from weather channel, Endocia Label server API (labels with USPS postage)
* Library of Classes: Framework
  + Developed as a package to be integrated (compiled) with component frameworks such as .NET or J2EE.
  + Examples: parse kit for MAC OS X apps (Scanners/parsers)
* Stand Alone Software Systems that are configured for use in a particular environment.
  + Often called COTS: “Commercial OFF the Shelf” systems
  + Example: PeopleSoft, HR management for companies

Modeling Diagrams

* Activity Diagram – Supplements the use case by providing a graphical representation of the flow of interaction within a specific scenario.
  + Uses flow chart-like symbols
    - Rounded rectangle – represents a specific system function/action.
    - Arrow – represents the flow of control from one function/action to another.
    - Diamond – represents a branching decision.
    - Solid Bar – represents the fork and join of parallel activities.
* Sequence Diagram – visualize temporal message ordering of a concrete scenario of a use case.
* State Diagram – Specify the abstract states of an object and the transitions between the states.
* Data flow diagrams – Control flow diagrams processing narratives.
  + Data Flow Diagram (DFD) is the diagrammatic form that is used.
  + Considered by many to be an ‘Old School’ approach, flow-oriented modeling continues to provide a view of the system that is unique-it should be used to supplement other analysis model elements.

Input 🡪 Computer Based System 🡪 Output

Elements of the Analysis Model

* Scenario-Based Elements
  + Use-case diagrams
  + Use cases-text
  + **Activity diagrams**
  + Swim lane diagrams
* Case-Based Elements
  + Class diagrams
  + Analysis packages
  + CRC Models
  + Collaboration Diagrams
* Behavioral Elements
  + **State Diagrams**
  + **Sequence Diagrams**

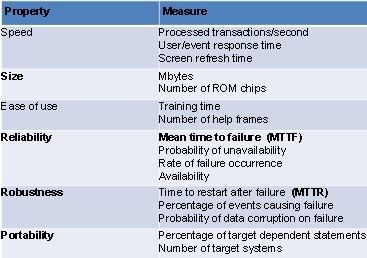
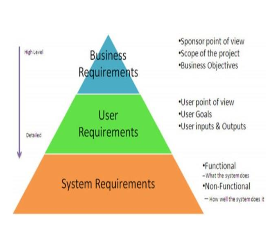
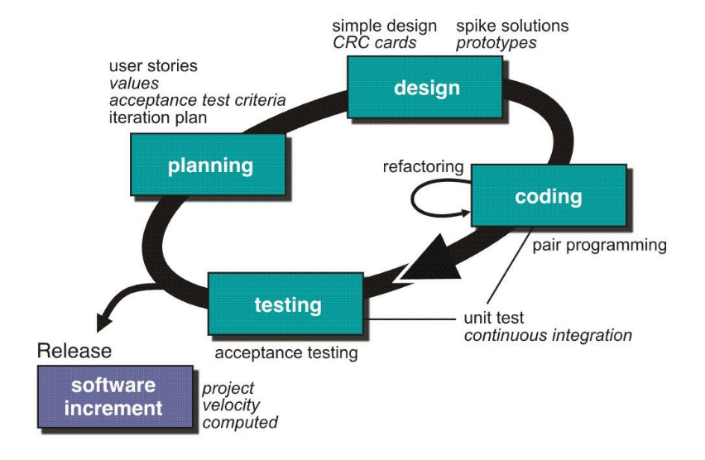
Structured Analysis

1. Flow-Oriented Elements
   * **Data-Flow Diagrams**
   * Control Flow Diagrams
   * Processing Narratives

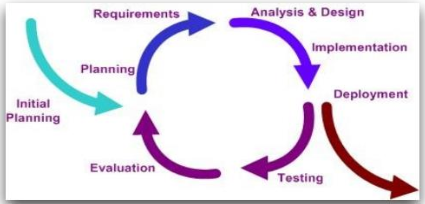
Alternate flows: The optional situations for the use case. What are the possibilities of what can go wrong? Which resources might be blocked, limited, unavailable? Which alternate flows are special, non-functional requirements – requirements?

1. Select the option that suits the Manifesto for Agile Software development
   * Individuals and interactions
   * Working software
   * Responding to change
   * **All of the mentioned**
2. Which one of the following is a functional requirement?
   * Maintainability
   * Portability (non-function)
   * Robustness (non-function)
   * Testability (non-function)
   * **None of the mentioned**
3. The essence of software engineering practice might be described as understand the problem. Plan a solution, carry out the plan, and examine the result for accuracy.
   * **True**
4. Analysis models depict software in which three representations
   * **Information, function, behavior**
5. Structure analysis focus only on the process that transforms the data
   * **False**
6. Which model in system modeling depicts the dynamic behavior of the system?
   * **Behavioral model**
7. Activity diagrams are used to model the processing of data.
   * **True**
8. Selection of a Life Cycle Model. Selection of a model is based on
   * Requirements
   * Development team
   * Users
   * Project type and associated risk
   * **All of the above**
9. Which of the following is not necessary to apply agility to a software process?
   * **Only essential work products are produced**
10. In extreme programming (XP). Programmers must complete a module before going home, taking a break, or moving on another task.
    * **False**

Requirements Engineering

1. The requirement Engineering (RE) is the most important phase of the Software Development Life Cycle (SDLC). Name and describe the main processes in requirements engineering.
   1. Elicitation – Gathering requirements
   2. Analysis – Understanding and modeling the desired behavior
   3. Specification – Defining what the system should do
   4. Development – Defining the organization of the system.
   5. Validation – Checking that the system does what the customer wants
   6. Evolution – Changing the system with response to the customer needs.

Iterative Process



* Reduces cost of accommodating changing customer requirements
  + Early versions are incomplete, so less rework to do.
  + May require no changes to current version (add to future version).
* It is easier to get customer feedback.
  + Users understand a working incremental release better than documents

from the specification or design phase.

* Does not need to be planned entirely up front.
* Early versions can implement the most important, urgent, or risky features

Cons:

* The process is not visible
* There’s less process documentation, so it’s difficult to measure progress.
* May not know how many more increments are required.
* Difficult to design and implement common facilities needed by all versions.
* System structure tends to degrade as new increments are added.
  + This makes the code more difficult to modify each time.
  + UNLESS time and money are spent on refactoring to improve the Software.
  + Refactoring: disciplined technique for restructuring an existing body of code,

altering its internal structure without changing its external behavior.

* Modifying a program to improve its structure, reduce its complexity, or make it easier to understand.