Software Life Cycle Processes

Objective:

- Establish a work plan to coordinate effectively a set of tasks.
- Improves software quality.
- Allows us to manage projects more easily.
- Status of projects is more easily tracked.
- Provides a baseline for improvement and measurement.

Build-and-Fix Model:

- Build a product without specs or any attempt at design
- Rework the product to satisfy the client
- It is not practical for product of reasonable size.
- Approach:

  - Build first version
  - Modify until customer is satisfied
  - Maintenance
Definition:

“A life cycle is the course of development changes through which a project passes from its inception as a project request to the mature state as characterized by a stable production environment.”

According to IEEE Std 12207 (ISO/IEC 12207), Standard for Information Technology -- Software life cycle processes:

- Establishes a common framework for software life cycle processes that can be referenced by the software industry.
- Lists processes that can be applied during the acquisition, supply, development, operation and maintenance of software and systems containing software.

Development Process

- Note that development often accounts for less than a third of the total project!
- Once a system has been built, the time spent on maintenance will be greater than the time spent on all other phases
  - Maintenance takes around 60% of total cost.
  - But note that for a very successful system (say, Unix, or Microsoft Word), maintenance can continue for a very long time.
- In fact, some describe developing software as the process of maintaining a blank sheet of paper!
Different Processes:

- **Problem identification**: someone identifies a problem, which needs addressing and might be susceptible to a computerized solution.

- **Requirements analysis**:  
  - It is much harder than people think.  
  - Required features that should be included in a computerized solution to provide a cost-effective solution to the problem.  
  - Capture what the customer wants.

- **Develop test plan**: Develop test procedures to determine if a completed product correctly provides all of required features.

- **Develop software design**:  
  - Analyze the required features to identify existing software that can be reused in this product.  
  - Changes needed in reused Components.  
  - Create high level design for new components  
  - For each component in high level design, develop detailed design.  
  - Also develop unit test plan for each component.
● **Built and Document:**

  ● Develop code for each module.

  ● Develop user documentation; help text, and operations manuals to support the use and operation of the system.

● **Validation:** Test each module

● **Integration and comprehensive testing:** Combine different modules into completed system and test the whole system.

● **Maintenance:** Once a system has been built, the time spent on maintenance will be greater than the time spent on all other phases.

● **Production:** Roll out the system to the users and monitor its performance.
Examples of life cycle models:

- Waterfall: development done in stages. Customer signs off on each stage before continuing to the next. This addresses the expense of changing requirements in later stages.

- Evolutionary: interleave activities until it's decided that any new features would cost more than justified.

- Formal transformation: produce formal, mathematical specification and transform this to create running system. Advantages: if transformations are correct, final system is guaranteed to meet its specification. Disadvantage: we can't apply these solutions to any reasonably sized system.

- Assembly from reusable components: all new systems are just constructed from existing components requires large library of components.

- Spiral Model: build system in phases; each phase is divided into stages including (informally) planning, review, risk analysis, construction, and evaluation.
The Waterfall Model

- Royce, 1970

- Consists of a set of sequential steps that a project progresses through.

- Each step must be completed before the project can progress to the next step.

- At the end of each step is some form of gateway, usually a formal review where that decision is made.

- There is no overlap between steps.

- Straightforward, simple to understand and use.

- Deliverables are frozen at the end of each phase and serve as the baseline for the following phases.

- You do not see the software until the end of the project.
• Approach:

Requirements analysis and specification ➔ Design and Specification ➔ Coding and Module Testing ➔ Integration and System Testing ➔ Delivery ➔ Maintenance
• Description of each waterfall step:

1. Requirements analysis and specifications:
   • Qualities required for the product: ease of use, performance, portability, etc.
   • Goals: an understandable, precise, complete, and consistent specification document which all people involved in the realization of the product can understand

2. Design and specification:
   • Partition system into modules
   • Establish overall system architecture and the relationship between different modules (i.e., IS_COMPOSED_OF, USES, etc.)
   • It should be straightforward to transform design into working program
   • Result: a design specification document

3. Coding and module testing:
   • Code and test the pieces
   • Module testing is the main control quality carried in this step.
4. **Integration and system testing:**
   - Part of the previous step in case of incremental development
   - Integrate modules
   - Test complete system: alpha testing

5. **Delivery**
   - Delivery for beta testing
   - Delivery for customer for production
   - Maintenance includes fixing errors, improving existing implementation, and adding new features

6. **Maintenance**
   - Cost of the maintenance is more than 60% of the total cost.
   - It consists of three categories:
     - Corrective maintenance: correction of any remaining errors
     - Adaptive maintenance: due to change in the environment
     - Perfective maintenance: improve the quality of the product by adding new features.

   - Pure waterfall is not practical:
     - Software product may not always be a linear process
• Often must freeze requirements in order to build system; this can result in an unusable (though correct) system.

• EVERY project involves some feedback.

• This model does not stress the anticipating changes: long and complex software products

• Relaxed Waterfall model:
• Incremental or Evolutionary Model:

• User given access to intermediate versions of the system and asked for new features.
• This method is probably the most commonly used one where the user is the developer or in small, very complex systems such as AI applications.
• Special skills are needed: this method works best with small teams of very skilled programmers who are adept at keeping track of lots of details simultaneously.
• In this model, the software product is engineered in a series of incremental builds.
• Each build consists of pieces of codes from various modules and satisfies a set of customer’s requirements.
• Each build is designed, implemented, tested, and integrated with the current system.
• Ex. Builds of an OS: scheduler, File management system, etc.
• The process stops when the product satisfies all specifications.
• Challenge: How many builds should a system have?
  • Varies from product to another
    • Few builds ➔ Build and fix approach
    • Too many builds ➔ high cost and very Little progress
• Approach:

Requirements analysis and specification

Design and Specification

For each build:
• Perform detailed design, implement, and integration.
• Test.
• Deliver to client

Maintenance

• Challenge: Integration of next build should be simple, straightforward, efficient, and cost effective.

• Developing relatively small systems where system can always be reimplemented from scratch with reasonable cost when maintenance gets out of hand
More risky incremental model:

- Specifications are done for each build separately.

- Major risk: Resulting builds may not fit together.
Boehm's Spiral Model

- The spiral model is the most generic of the models.
- Most life cycle models can be derived as special cases of the spiral model.
- Uses risk management approach to software development.
- Software development ➔ Risk
  - Key person can resign
  - Manufacturer of hardware can go bankrupt
- Software developers ➔ Try to minimize risks using prototypes
- Advantages: (http://www.levela.com/software_life_cycles_swdoc.htm)
  - Defers elaboration of low risk software elements
  - Incorporates prototyping as a risk reduction strategy
  - Gives an early focus to reusable software
  - Accommodates life-cycle evolution, growth, and requirement changes
  - Incorporates software quality objectives into the product
  - Focus on early error detection and design flaws
  - Sets completion criteria for each project activity to answer the question: "How much is enough?"
  - Uses identical approaches for development and maintenance
  - Can be used for hardware-software system development

- Definitions:
  According to Boehm: “Risk management is a discipline whose objectives are to identify, address, and eliminate software risk items before they become either threats to successful software operation or a major source of expensive software rework.”

- Spiral model focuses on minimizing high-risk problems by careful process design and prototyping.
- Unlike the linear waterfall model, spiral model is cyclic:

- Each cycle (loop) of the spiral consists of 4 phases and each phase is represented by a quadrant of the Cartesian diagram:

1. Identify the objectives, alternatives, and constraints:
   - Identify objects for a phase
   - Identify constraints
   - Identify risks
   - Identify alternative courses of action
   - Establish a strategy to achieve these objectives
2. Evaluate alternatives, identify, and resolve risks:
   - Analyze the strategy defined in (1) from the viewpoint of risk.
   - If risks cannot be resolved, the project may be terminated or it may be scaled to a smaller size.
   - Take steps to reduce risk (e.g.: develop prototype)
   - Go to (3) if all risks are successfully resolved.

3. Develop and verify the next-level product:
   - This corresponds to the pure waterfall model.

4. Plan next phase:
   - Review the results of the previous phases and plan for the next iteration of the spiral.
   - Commitment: Continue with spiral?
• **Careful**: the division of a spiral into 4 phases does not mean you'll spend equal time on each part!

• **Restrictions:**
  • Internal development of large-scale software: customers and developers belong to the same organization.
  • Spiral model can only be applied to large-scale software product.
  • It does not make sense to perform risk analysis if the cost of performing the risk analysis is comparable to the cost of the whole product.