Software Life Cycle Models

Acknowledgement: Fundamentals of Software Engineering (**Prof. Rajib Mall**)

Software Life Cycle

- Software life cycle (or software process):
  - Series of identifiable stages that a software product undergoes during its life time:
    - Feasibility study
    - Requirements analysis and specification,
    - Design,
    - Coding,
    - Testing
    - maintenance.

Life Cycle Model

- A software life cycle **model** (or process model):
  - a descriptive and diagrammatic model of software life cycle
  - identifies all the activities required for product development,
  - establishes a precedence ordering among the different activities,
  - Divides life cycle into phases.

Life Cycle Model (CONT.)

- Several different activities may be carried out in each life cycle phase.
  - For example, the design stage might consist of:
    - structured analysis activity followed by
    - structured design activity.

Why Model Life Cycle ?

- A written description:
  - Forms a common understanding of activities among the software developers.
  - Helps in identifying inconsistencies, redundancies, and omissions in the development process.

Life Cycle Model (CONT.)

- The development team must identify a suitable life cycle model:
  - and then **adhere to it**.
  - Primary advantage of adhering to a life cycle model:
    - Helps development of software in a systematic and disciplined manner.

Life Cycle Model (CONT.)

- When a program is developed by a single programmer ---
  - he has the freedom to decide his exact steps.

Life Cycle Model (CONT.)

- When a **software product** is being developed by a **team**:
  - there must be a precise understanding among team members as to ***when to do what***,
  - otherwise it would lead to chaos and project *failure*.

Life Cycle Model (CONT.)

A software project will never succeed if:

- one engineer starts writing code,
- another concentrates on writing the test document first,
- yet another engineer first defines the file structure
- ….

Life Cycle Model (CONT.)

A life cycle model:

- defines **entry** and **exit** criteria for every phase.

- A phase is considered to be **complete**:
  - only when all its **exit** criteria are satisfied.

Life Cycle Model (CONT.)

- The phase **exit** criteria for the software requirements specification phase:
  - Software Requirements Specification (SRS) document is complete, reviewed, and approved by the customer.

- A phase can **start**:
  - only if its ***phase-entry criteria*** have been satisfied.

Life Cycle Model (CONT.)

- It becomes easier for software project managers:
  - to **monitor the progress** of the project.

Life Cycle Model (CONT.)

- When a life cycle model is adhered to,
  - the project manager can at any time fairly accurately tell,
    - at which stage (e.g., design, code, test, etc.) of the project is.
  - Otherwise, it becomes very difficult to track the progress of the project
    - the project manager would have to depend on the guesses of the team members.

Life Cycle Model (CONT.)

- This usually leads to a problem:
  - known as the **99% complete** syndrome.

Life Cycle Model (CONT.)

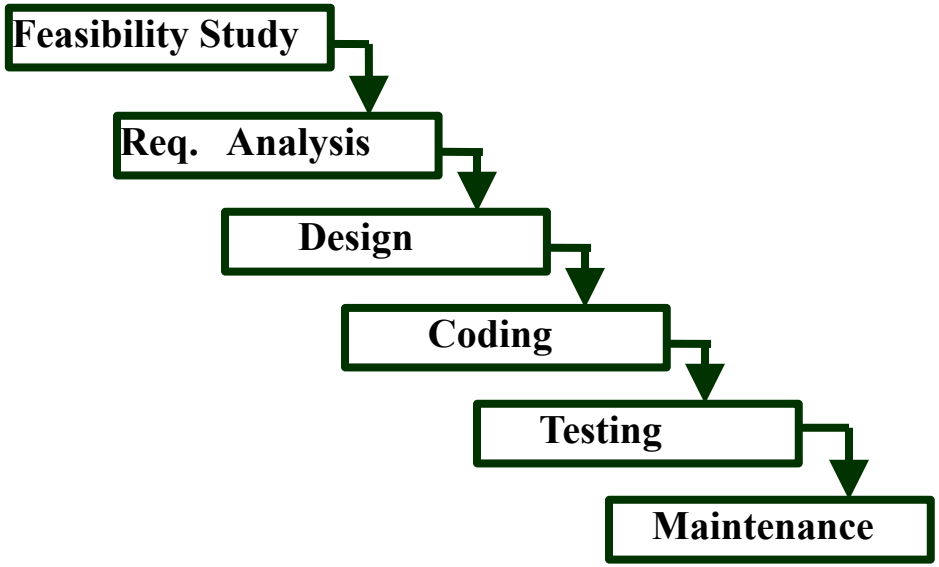
- Many life cycle models have been proposed.
- We will confine our attention to a few important and commonly used models.
  - Classical waterfall model
  - Iterative waterfall,
  - Evolutionary,
  - Prototyping, and
  - Spiral model

Classical Waterfall Model

Classical waterfall model divides life cycle into phases:

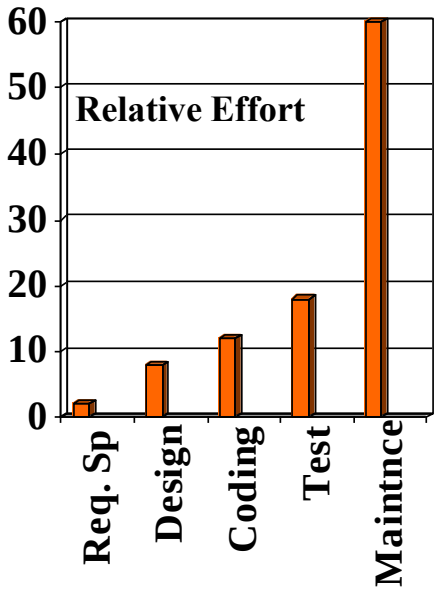
- feasibility study,
- requirements analysis and specification,
- design,
- coding and unit testing,
- integration and system testing,
- maintenance.

Classical Waterfall Model



Relative Effort for Phases

- Phases between feasibility study and testing
  - known as development phases.
- Among all life cycle phases
  - maintenance phase consumes maximum effort.
- Among development phases,
  - testing phase consumes the maximum effort.



Classical Waterfall Model (CONT.)

- Most organizations usually define:
  - standards on the outputs (deliverables) produced at the end of every phase
  - **entry** and **exit** **criteria** for every phase.

- They also prescribe specific methodologies for:
  - specification,
  - design,
  - testing,
  - project management, etc.

Classical Waterfall Model (CONT.)

- The guidelines and methodologies of an organization:
  - called the organization's software development methodology.
- Software development organizations:
  - expect fresh engineers to master the organization's software development methodology.

Feasibility Study

Main aim of feasibility study: determine whether developing the product

- **financially** worthwhile
- **technically** feasible.

First roughly understand what the customer wants:

- different data which would be input to the system,
- processing needed on these data,
- output data to be produced by the system,
- various constraints on the behavior of the system.

Activities during Feasibility Study

- Work out an overall understanding of the problem.
- Formulate different solution strategies.
- Examine alternate solution strategies in terms of:
  - resources required,
  - cost of development, and
  - development time.

Activities during Feasibility Study

- Perform a **cost/benefit** analysis:
  - to determine which solution is the best.
  - you may determine that none of the solutions is feasible due to:
    - high cost,
    - resource constraints,
    - technical reasons.

Requirements Analysis and Specification

- Aim of this phase:
  - understand the exact requirements of the customer,
  - document them properly.

- Consists of two distinct activities:
  - requirements **gathering** and **analysis**
  - requirements **specification**.

Goals of Requirements Analysis

- **Collect all related data** from the customer:
  - analyze the collected data to clearly understand what the customer wants,
  - find out any inconsistencies and incompleteness in the requirements,
  - resolve all inconsistencies and incompleteness.

Requirements Gathering

- Gathering relevant data:
  - usually collected from the end-users through ***interviews*** and ***discussions***.
  - For example, for a business accounting software:
    - interview all the accountants of the organization to find out their requirements.

Requirements Analysis (CONT.)

- The data you initially collect from the users:
  - would usually contain several contradictions and ambiguities:
  - each user typically has only a partial and incomplete view of the system.

Requirements Analysis (CONT.)

- Ambiguities and contradictions:
  - must be identified
  - resolved by discussions with the customers.

- Next, requirements are organized:
  - into a Software Requirements Specification (SRS) document.

Requirements Analysis (CONT.)

- Engineers doing requirements analysis and specification:
  - are designated as analysts.

Design

- Design phase transforms requirements specification:
  - into a form suitable for implementation in some programming language.

Design

- In technical terms:
  - during design phase, software architecture is derived from the SRS document.
- Two design approaches:
  - traditional approach,
  - object oriented approach.

Traditional Design Approach

- Consists of two activities:
  - Structured analysis
  - Structured design

Structured Analysis Activity

- **Identify all the functions** to be performed.
- Identify **data flow among the functions**.
- **Decompose** each function recursively into sub-functions.
  - Identify data flow among the sub-functions as well.

Structured Analysis (CONT.)

- Carried out using Data flow diagrams (DFDs).

- After structured analysis, carry out structured design:
  - architectural design (or high-level design)
  - detailed design (or low-level design).

Structured Design

- High-level design:
  - decompose the system into *modules*,
  - represent invocation relationships among the modules.

- Detailed design:
  - different modules designed in greater detail:
    - data structures and algorithms for each module are designed.

Object Oriented Design

- First identify various objects (real world entities) occurring in the problem:
  - identify the relationships among the objects.
  - For example, the objects in a pay-roll software may be:
    - employees,
    - managers,
    - pay-roll register,
    - Departments, etc.

Object Oriented Design (CONT.)

- Object structure
  - further refined to obtain the detailed design.
- OOD has several advantages:
  - lower development effort,
  - lower development time,
  - better maintainability.

Implementation

- Purpose of **implementation phase** (aka coding and unit testing phase):
  - **translate software design into source code**.

Implementation

- During the implementation phase:
  - each module of the design is **coded**,
  - each module is **unit tested**
    - tested independently as a stand alone unit, and debugged,
  - each module is **documented**.

Implementation (CONT.)

- The purpose of unit testing:
  - test if individual modules work correctly.
- The end product of implementation phase:
  - a set of program modules that have been tested individually.

Integration and System Testing

- Different modules are integrated in a planned manner:
  - modules are almost never integrated in one shot.
  - Normally integration is carried out through a number of steps.
- During each integration step,
  - the partially integrated system is tested.

System Testing

- After all the modules have been successfully integrated and tested:
  - system testing is carried out.

- Goal of system testing:
  - ensure that the developed system functions according to its requirements as specified in the SRS document.

Maintenance

Maintenance of any software product:

- requires much more effort than the effort to develop the product itself.
- development effort to maintenance effort is typically **40:60**.

Why do we need maintenance?

Maintenance (CONT.)

- **Corrective maintenance**:
  - Correct errors which were not discovered during the product development phases.
- **Perfective maintenance**:
  - Improve implementation of the system
  - enhance functionalities of the system.
- **Adaptive maintenance**:
  - Port software to a new environment,
    - e.g. to a new computer or to a new operating system.

Iterative Waterfall Model

- Classical waterfall model is idealistic:
  - assumes that no **defect** is introduced during any development activity
  - in practice:
    - defects may get introduced in almost every phase of the life cycle.

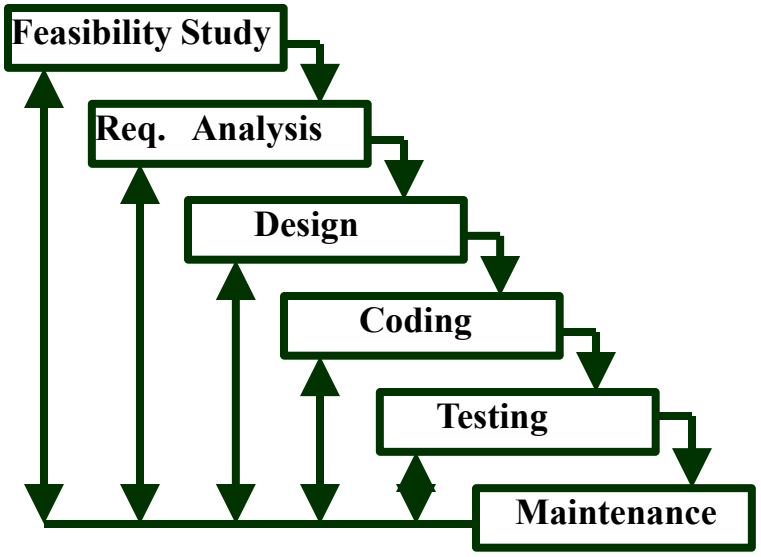
Iterative Waterfall Model (CONT.)

- Defects usually get detected much later in the life cycle:
  - For example, a design defect might go unnoticed till the coding or testing phase.

Iterative Waterfall Model (CONT.)

- Once a defect is detected:
  - we need to go back to the phase where it was introduced
  - redo some of the work done during **that and all subsequent phases**.
- Therefore we need **feedback paths** in the classical waterfall model.

Iterative Waterfall Model (CONT.)



Iterative Waterfall Model (CONT.)

- Errors should be detected
  - in the same phase in which they are introduced.
- For example:
  - if a design problem is detected in the design phase itself,
    - the problem can be taken care of much more easily
    - than say if it is identified at the end of the integration and system testing phase.

Phase containment of errors

- Reason: rework must be carried out not only to the design but also to code and test phases.

- The principle of detecting errors as close to its point of introduction as possible:
  - is known as **phase containment of errors.**

- Iterative waterfall model is by far the most widely used model.
  - Almost every other model is derived from the waterfall model.

Prototyping Model

Prototyping Model

- Before starting actual development,
  - a working prototype of the system should first be built.

- A prototype is a **toy implementation** of a system:
  - limited functional capabilities,
  - low reliability,
  - inefficient performance.

Reasons for developing a prototype

- **Illustrate to the customer**:
  - input data formats, messages, reports, or interactive dialogs.

- **Examine technical issues** associated with product development:
  - Often major design decisions depend on issues like:
    - response time of a hardware controller,
    - efficiency of a sorting algorithm, etc.

Prototyping Model (CONT.)

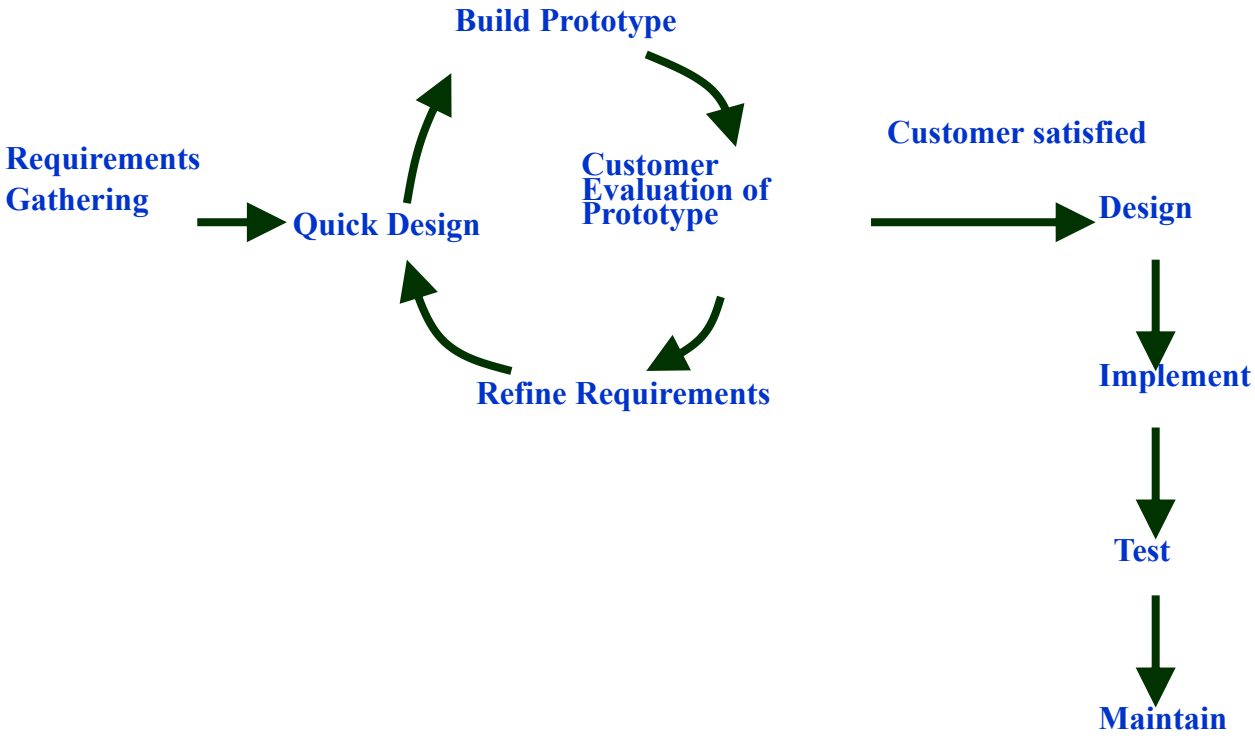
- Start with ***approximate*** requirements.
- Carry out a quick design.
- **Prototype** model is built using several short-cuts:
  - Short-cuts might involve using inefficient, inaccurate, or dummy functions.
    - A function may use a table look-up rather than performing the actual computations.

Prototyping Model (CONT.)

- The developed prototype is submitted to the customer for his evaluation:
  - Based on the user feedback, **requirements are refined**.
  - This cycle continues until the user approves the prototype.

- The actual system is developed using the classical waterfall approach.

Prototyping Model (CONT.)



Prototyping Model (CONT.)

Final working prototype (with all user feedbacks incorporated) serves as an ***animated*** requirements specification.

- Design and code for the prototype is usually thrown away:
  - However, the experience gathered from developing the prototype helps a great deal while developing the actual product.

Prototyping Model (CONT.)

- Even though construction of a working prototype model involves additional cost --- overall development cost might be lower for:
  - systems with unclear user requirements,
  - systems with unresolved technical issues.

- **Many user requirements get properly defined and technical issues get resolved**:
  - these would have appeared later as change requests and resulted in incurring **massive** redesign costs.

Evolutionary Model

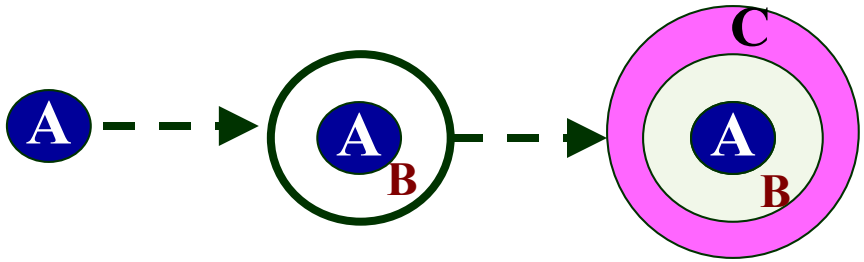
- Evolutionary model (aka successive versions or incremental model):
  - The system is broken down into several modules which can be incrementally implemented and delivered.

- **First develop the core modules** of the system.
- The initial product skeleton is refined into increasing levels of capability:
  - by adding new functionalities in successive versions.

Evolutionary Model (CONT.)

- Successive version of the product:
  - functioning systems capable of performing some useful work.
  - A new release may include new functionality:
    - also existing functionality in the current release might have been enhanced.

Evolutionary Model (CONT.)



Advantages of Evolutionary Model

- Users get a chance to **experiment with a partially developed system**:
  - much before the full working version is released,
- Helps **finding exact user requirements**:
  - much before fully working system is developed.
- **Core modules get tested thoroughly**:
  - reduces chances of errors in final product.

Disadvantages of Evolutionary Model

- Often, difficult to subdivide problems into functional units:
  - which can be incrementally implemented and delivered.
  - evolutionary model is useful for very large problems,
    - where it is easier to find modules for incremental implementation.

Evolutionary Model with Iteration

- Many organizations use a combination of ***iterative*** and ***incremental*** development:
  - a new release may include new functionality
  - existing functionality from the current release may also have been modified.

Evolutionary Model with iteration

- Several advantages:
  - **Training can start** on an earlier release
    - customer feedback taken into account
  - **Markets can be created**:
    - for functionality that has never been offered.
  - **Frequent releases** allow developers to **fix unanticipated problems** quickly.

Spiral Model

- Proposed by Boehm in 1988.
- Each loop of the spiral represents a phase of the software process:
  - the innermost loop might be concerned with system feasibility,
  - the next loop with system requirements definition,
  - the next one with system design, and so on.
- There are no fixed phases in this model.

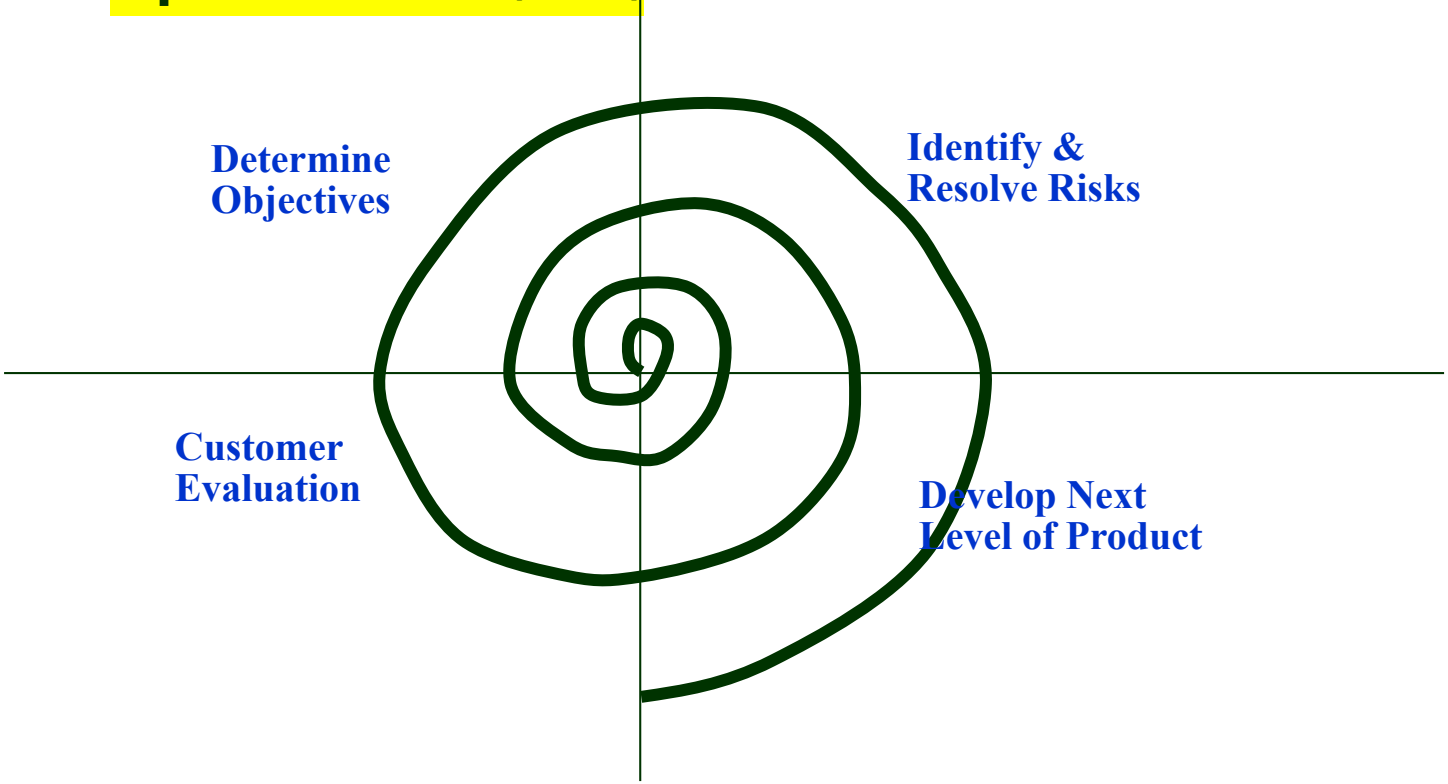
Spiral Model (CONT.)

- The team must decide:
  - how to structure the project into phases.

- Start work using some generic model:
  - add extra phases
    - for specific projects or when problems are identified during a project

- Each **loop in the spiral** is split into four sectors (quadrants).

Spiral Model (CONT.)



Objective Setting (First Quadrant)

- Identify objectives of the phase,
- Examine the risks associated with these objectives.
  - Risk:
    - any adverse circumstance that might hamper successful completion of a software project.
- Find alternate solutions possible.

Risk Assessment and Reduction (Second Quadrant)

- For each identified project risk,
  - a detailed analysis is carried out.
- Steps are taken to reduce the risk.

- For example, if there is a risk that the requirements are inappropriate:
  - a prototype system may be developed.

Spiral Model (CONT.)

- Development and Validation (Third quadrant):
  - develop and validate the next level of the product.
- Review and Planning (Fourth quadrant):
  - review the results achieved so far with the customer and plan the next iteration around the spiral.
- With each iteration around the spiral:
  - progressively more complete version of the software gets built.

Comparison of Different Life Cycle Models

- Iterative waterfall model
  - most widely used model.
  - But, suitable **only for well-understood problems**.

- Prototype model is suitable for projects not well understood:
  - user requirements
  - technical aspects

Comparison of Different Life Cycle Models
(CONT.)

- Evolutionary model is **suitable for large problems**:
  - can be **decomposed** into a set of modules that can be **incrementally** implemented,
  - incremental delivery of the system is acceptable to the customer.

- The spiral model:
  - suitable for development of **technically challenging** software products that are subject to several kinds of risks.