When you build a product or system, it's important to go through a series of predictable steps—a road map that helps you to create a timely and high-quality result. The road map that you follow is called a '***software process***'.

2.1-A Layered Technology

2.1.1-Quality, Process, Methods, and Tools

Software engineering is a layered technology. Most engineering approaches (including software engineering) must rest on an organizational commitment to quality. The bedrock that supports software engineering is a '***quality focus***' layer.

-Quality: a product should meet its specification. This is problematical for software systems. There is a tension between customer quality requirements (efficiency, reliability, etc.), developer quality requirements (maintainability, reusability, etc.), users (usability, efficiency, etc.), and etc. But note:

→Some quality requirements are difficult to specify in an unambiguous way.

→Software specifications are usually incomplete and often inconsistent.

-Process: The foundation for software engineering is the '***process***' layer. Software engineering process is the glue that holds the technology together and enables rational and timely development of computer software. The work products are produced, milestones are established, quality is ensured, and changes are properly managed.

-Methods: Software engineering ***methods*** provides the technical how-to's for building software. Methods encompass a broad array of tasks that include the requirements analysis, design, program construction, testing, and support.

-Tools: Software engineering ***tools*** provide automated or semi-automated supports for the process and the methods. When the tools are integrated so that the information created by one tool can be used by another, a system for the support of software development, called ***computer-aided software engineering (CASE)***. CASE combine software, hardware, and software engineering database.

2.1.2-A Generic View of Software Engineering

Engineering is the analysis, design, construction, verification, and management of technical entities. Regardless of entities to be engineered, the following questions must be asked and answered:

- . What is the problem to be solved?
- . What characteristics of entity are used to solve the problem?
- . How will the entity (and the solution) be realized?
- . How will the entity be constructed?
- . What approach will be used to uncover the errors that were made in the design and construction of the entity?
- . How will the entity be supported over the long term, when the ***corrections***, ***adaptations***, and ***enhancements*** are requested by the user of the entity?

Software is engineered by applying three distinct phases that focus on ***definition***, ***development***, and ***support***.

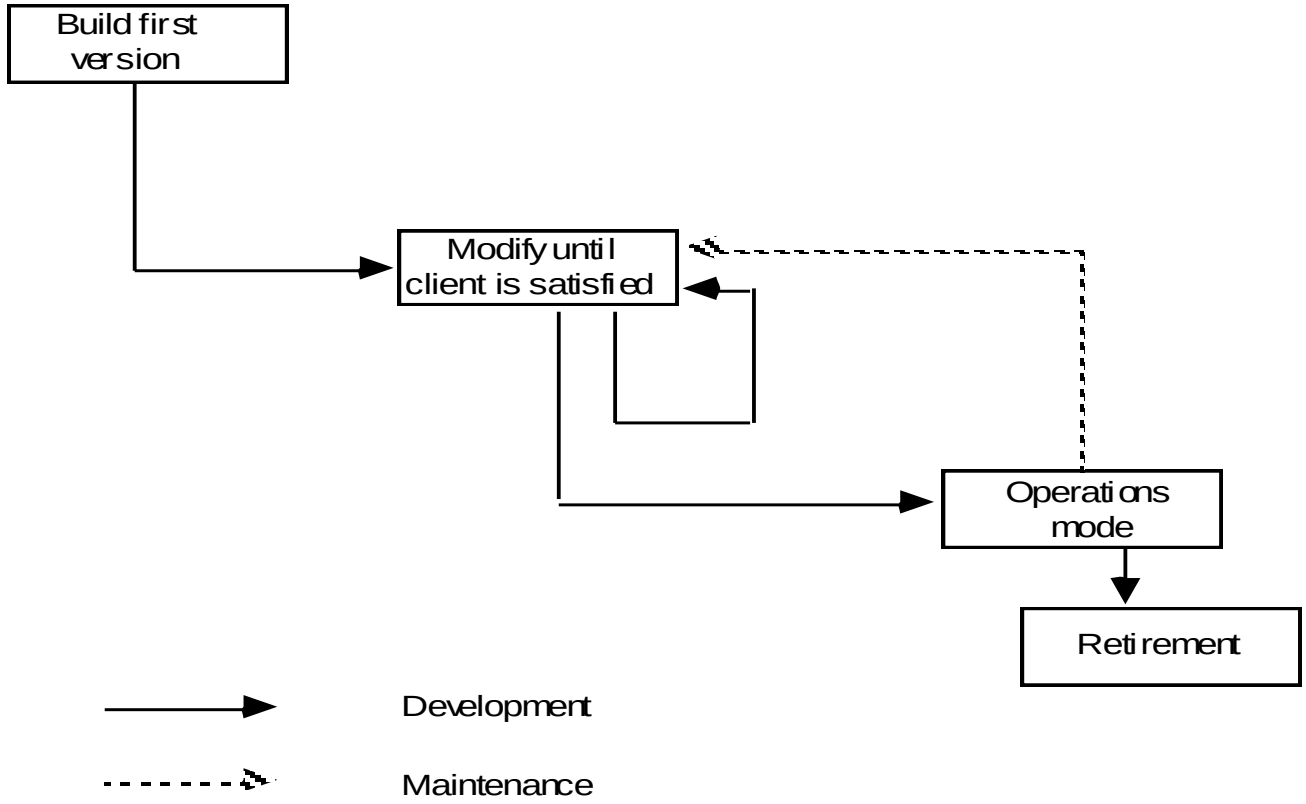
- The **definition phase** focuses on ***what***.
- The **development phase** focuses on ***how***.
- The **support phase** focuses on ***change***. The changes due to enhancements by changing customer requirements. The support phase reapplies the steps of definition and development phases. Four types of change are encountered during the support phase:
  - **. Correction.**
  - **. Adaptation.**
  - **. Enhancement.**
  - **. Prevention.**

2.2-Life Cycle

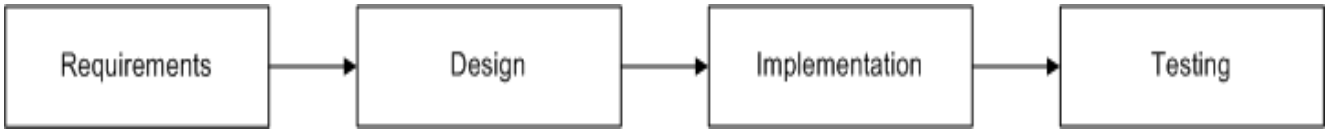
A Life Cycle shows how a living thing borns, grows, lives, and dies. The stages from birth to death.

2.2.1-SW Life Cycle Model

Software life cycle model is the stages of development that a software development goes through. The following figure shows the stages of software development.



Software life cycle models describe phases of the software cycle and the order in which those phases are executed. There are tons of models, and many companies adopt their own, but all have very similar patterns. The general, basic model is shown below:



Each phase produces deliverables required by the next phase in the life cycle. Requirements are translated into design. Code is produced during implementation that is driven by the design. Testing verifies the deliverable of the implementation phase against requirements.

2.2.2-New Ideal of SW Life Cycle Model

→It is a Build-and-fix (Code & Fix):

- without specs or design
- modify until customer is satisfied

→Difficulties:

- Problems to be solved were becoming more complex; harder to visualize without a lot more advance work in the thinking or design stage.
- Managers and users were becoming disenchanted with late delivery dates of promised code.
- Users were dissatisfied with the quality of the delivered products.
- Code was expensive to fix—poor preparation for testing and modification.

→Problems:

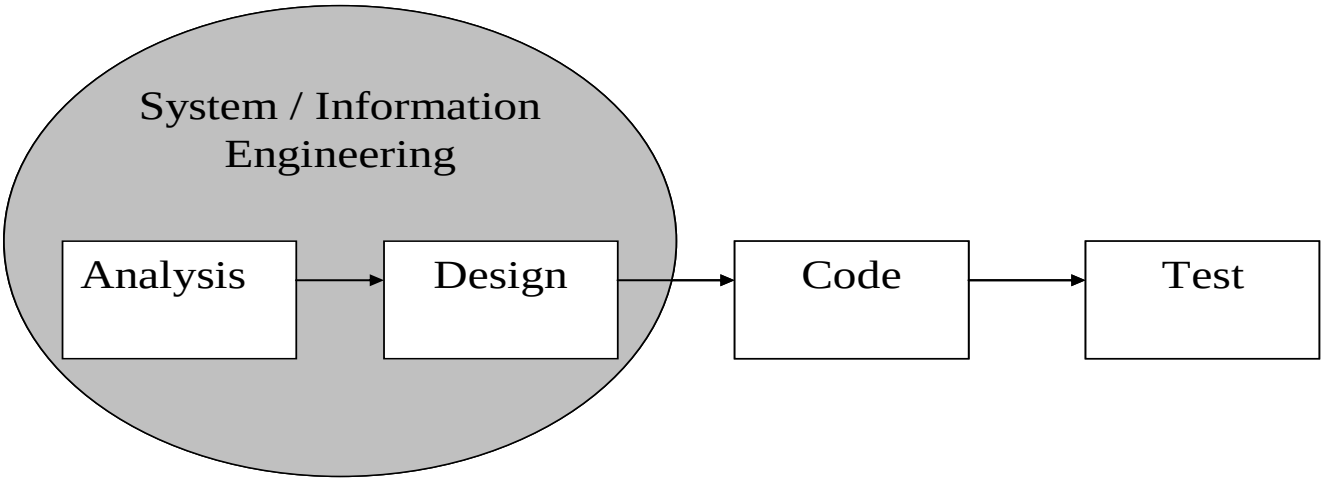
- Limited viewpoint
- User requirements that are missed or misinterpreted
- Delivery of a product the user does not want

- Cost overruns

2.3-The Linear Sequential Model

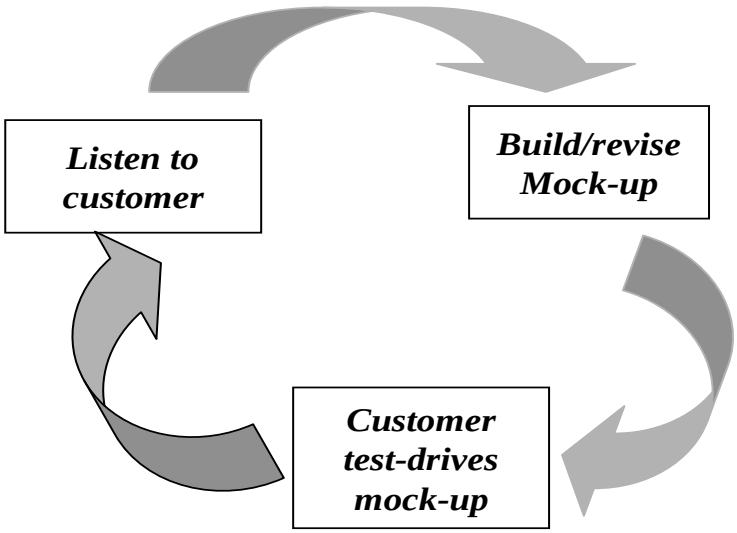
This is a software process model that involves a systematic progression through **analysis**, **design**, **coding**, **testing** and **maintenance** phases. It is also referred to as the "**waterfall model**".
Also known as the classic life cycle or waterfall model, it suggests a systematic, sequential approach to software development. Problems with this approach are:

- Real projects rarely follow the sequential flow and changes can cause confusion.
- This model has difficulty accommodating requirements change
- The customer will not see a working version until the project is nearly complete
- Developers are often blocked unnecessarily, due to previous tasks not being done



2.4-The Prototyping Model

2.4.1-Prototyping Model



+Advantages:

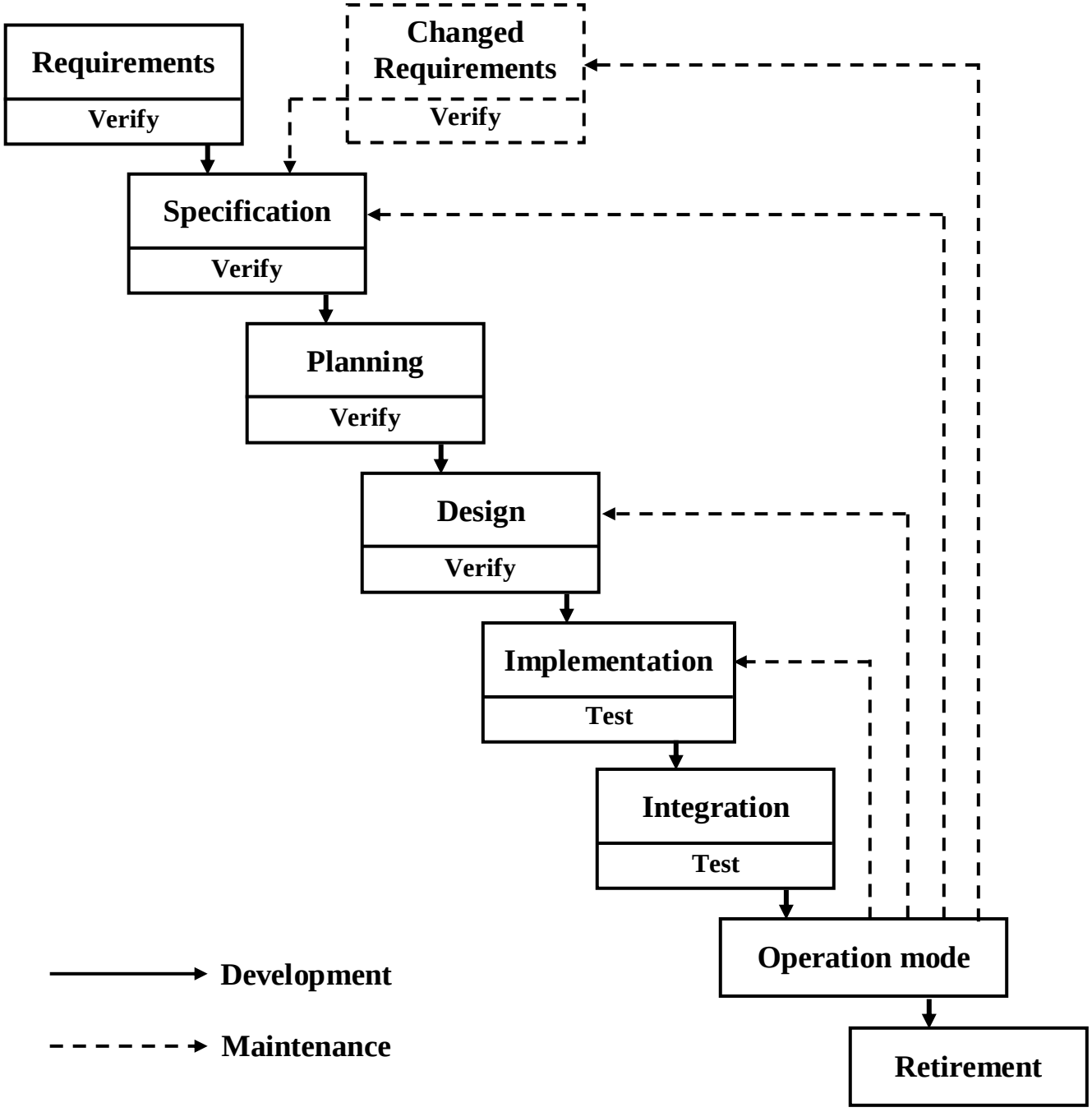
- Easy and quick to identify customer requirements
- Customers can validate the prototype at the earlier stage and provide their inputs and feedback
- Good to deal with the following cases:
  - → Customer can not provide the detailed requirements
  - → Very complicated system-user interactions
  - → Use new technologies, hardware and algorithms

→ Develop new domain application systems

Problems:

-The prototype can serve as "**the first system**".
-Developers usually attempt to develop the product based on the prototype.
-Developers often make implementation compromises in order to get a prototyping that is working quickly.
-Customers may be unaware that the prototype is not a product, which is held with.

2.4.2-Rapid Prototyping Model



A rapid prototype is a working model that is functionally equivalent to a subset of the product.

(1) Because the working prototype has been validated through interaction with the client, the resulting specification will be correct. Therefore a major strength of this model is that the development process is essential linear with little or no feedback loops.
(2) In specification, planning and design, verification is needed. In implementation and integration, testing is needed.
(3) An essential aspect of a rapid prototype is in the word ***rapid***.

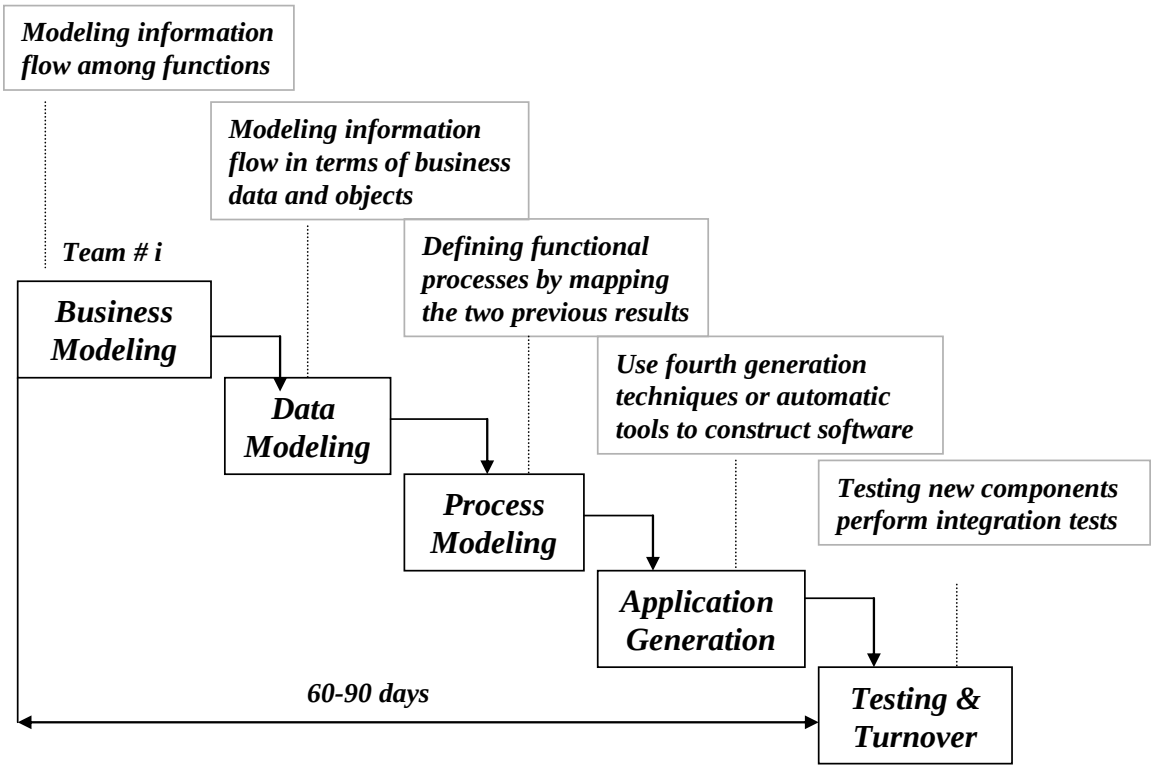
(4) We can combine waterfall and prototyping by using rapid prototyping to find out the client's requirements.

2.5-The RAD Model

Rapid Application Development (RAD) is a linear sequential software development process model that emphasizes an extremely short development cycle.
- A "high-speed" adaptation of linear sequential model
- Component-based construction
- Effective when requirements are well understood and project scope is constrained.

Advantages:
- Short development time
- Cost reduction due to software reuse and component-based construction

Problems:
- For large, but scalable projects, RAD requires sufficient resources.
- RAD requires developers and customers who are committed to the schedule.
- Constructed software is project-specific, and may not be well modularized.
- Its quality depends on the quality of existing components.
- Not appropriate projects with high technical risk and new technologies.



2.6-The Evolutionary Process Model

2.6.1- Incremental Model

A **build** consists of code pieces from various modules interacting to provide a specific functionality.

(1) Each build is designed, coded and integrated into the software structure that is tested as a whole.

(2) The incremental model, something will be working within weeks, with the rest delivered incrementally.

(3) Product can be introduced into the client's organization gradually.

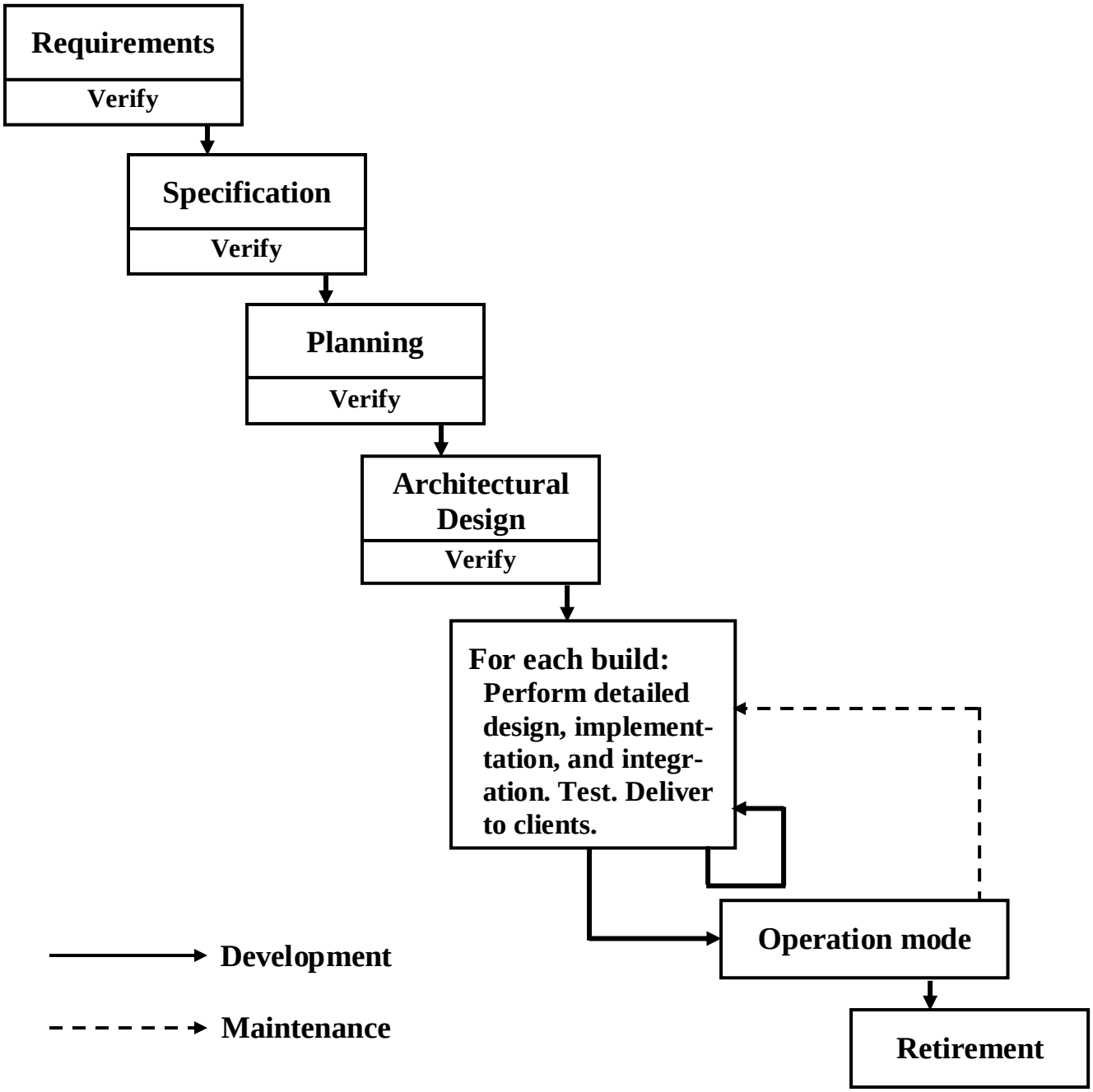
(4) Problems of integration, changes in requirements, "**shooting at a moving target**".

Advantages
- Generates working software quickly and early during the software life cycle.
- More flexible – less costly to change scope and requirements.

- Easier to test and debug during a smaller iteration.
- Easier to manage risk because risky pieces are identified and handled during its iteration.
- Each iteration is an easily managed milestone.

Disadvantages

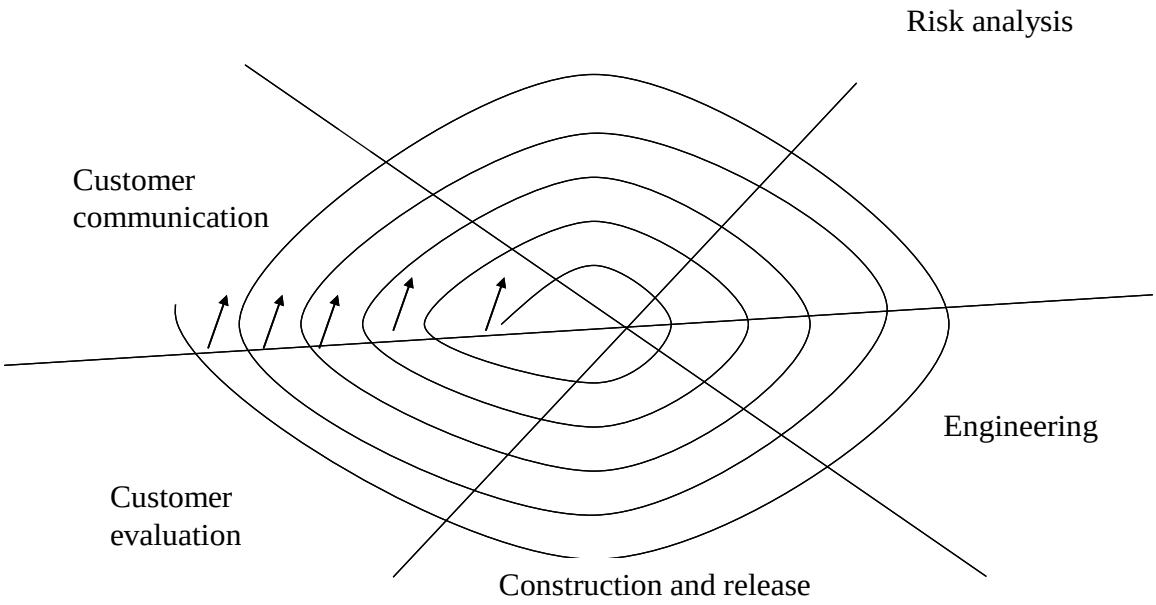
- Each phase of an iteration is rigid.
- Problems may arise pertaining to system architecture because not all requirements are gathered up front for the entire software life cycle.



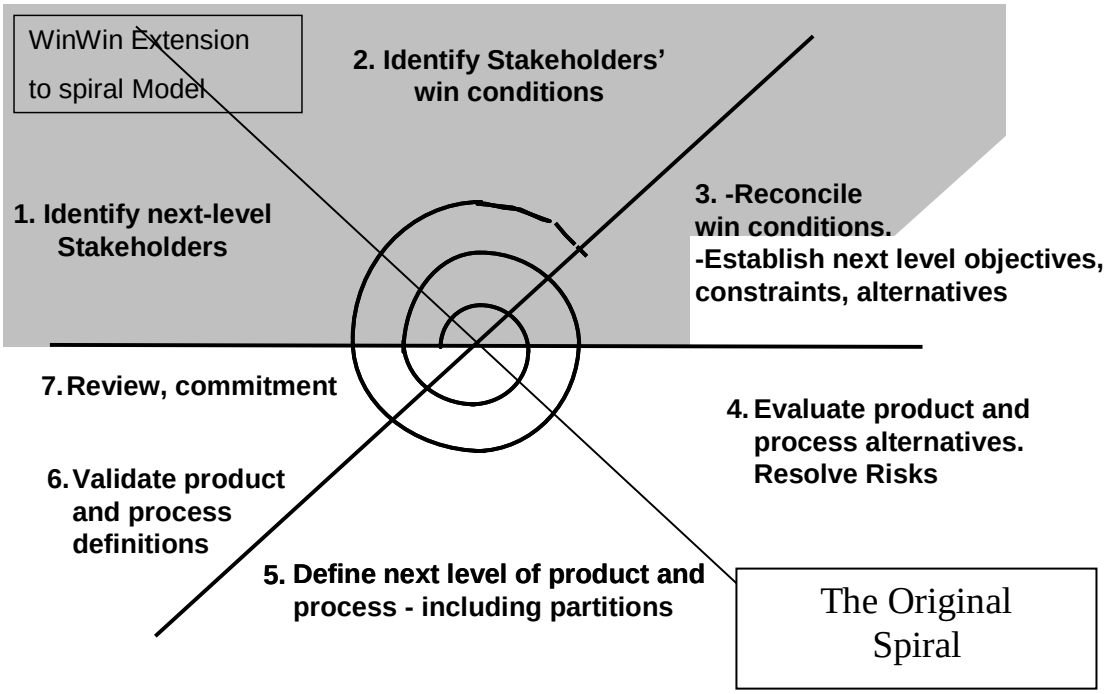
2.6.2-The Spiral Model

- Boehm's (1988) spiral model couples the iterative nature of prototyping with the controlled and systematic aspects of the linear sequential model.
- Software is developed in a series of incremental releases.
- During the early releases, there may be just a paper model, but the system becomes increasingly more complete.
- There are a number of framework activities (Customer communication, Planning, Risk analysis, Engineering, Construction and release, and Customer evaluation).

- Unlike any of the other models, this model keeps revisiting the system throughout its lifetime.



2.6.3-The WinWin Spiral Model



→Where do objectives, constraints, alternatives come from?
-WinWin extensions

→Lack of intermediate milestones

→Need to avoid model clashes, provide more specific guidance

→Adds three activities to the front end of each spiral cycle:

- Identify the system or subsystem's key stakeholders
- Identify the stakeholders' win conditions for the system or subsystem
- Negotiate win-win reconciliations of the stakeholders' win conditions

→The additional activities account for where the elaborated objectives, constraints, and alternatives come from (a noted difficulty with the spiral model as originally proposed). This more clearly identifies the rationale involved in negotiating the "win" conditions for the product.