UNIT-I:

Conventional Software Management: The Waterfall Model, Conventional Software Management Performance. Evolution of Software Economics: Software Economics, Pragmatic Software Cost Estimation.

CONVENTIONAL SOFTWARE MANAGEMENT:

- The best thing about the software is its flexibility: It can be programmed to do almost anything.
- The worst thing about the software is also its flexibility The almost anything" characteristic has made it difficult to Plan, monitor, and control software development-
- In the mid-1990s, at least three important analyses of the state of the Software Engineering industry were performed.

All three analyses reached the same general conclusion The Success rate for Software Project is very low-

They can be summarized as follows:

1. Software development is still highly unpredictable only about 10% of software Projects are delivered success. -fully within initial budget and Schedule estimates.

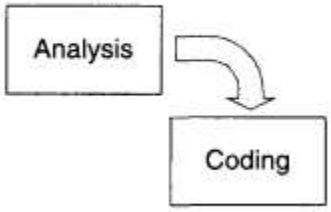
2. Management discipline is more of a discriminator in Success or fat lure than are technology advances."

3. The level of software scrap and rework is indicative of an immature Processors

The waterfall Model:

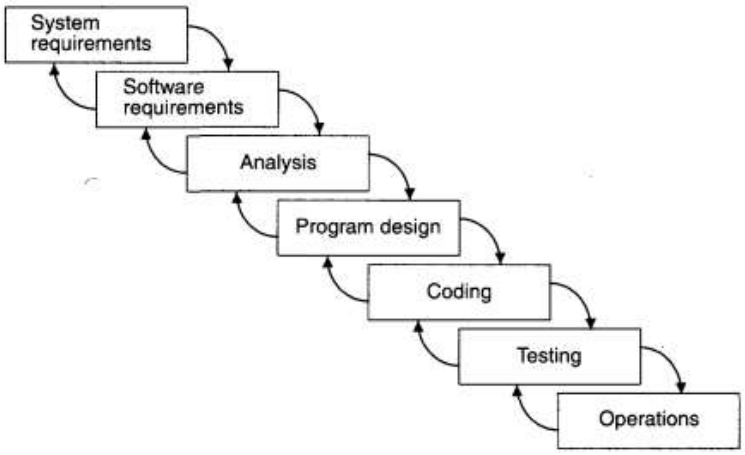
- The waterfall model is also called as linear sequential model or classic life cycle model.
- This section examines and critiques the water fall theory, then looks at how most of the industry has practices the conventional software process.
- There are two essential steps common to the development of computer programs: analysis and coding.
- In order to manage and control software development including system requirements definition, software requirements definition, program design, and testing. These steps supplement the analysis and coding steps.

Waterfall Model Part 1 : The two basic steps to building a program



Analysis and coding both involve creative work that directly contributes to the usefulness of the end product.

Waterfall Model Part 2 : The large-scale system approach



- The following are 5 improvements to the basic water fall process to eliminate the development risks.

1) Program design comes first:

- Insert a preliminary program design phase between the software requirements generation phase and the analysis phase.
- The insufficient resources and the design limitations are then identified in the early stages
- Begin the design process with program designers, not analysts or programmers.
- Allocate processing functions, design the database, allocate execution time, define interfaces and processing modes with the operating system, describe input and output processing, and define preliminary operating procedures.
- Write an overview document that is understandable, informative, and correct

2) Document the design:

- We know that document design is required a lot for software programs
- Each designer must communicate with interfacing designers, managers, and possibly customers.
- During early phases, the documentation is the design.

- The real value of documentation is to support later changes by a separate test team, a separate maintenance team, and operations personnel who are not software literate.

3) Do it twice if possible:

- The software delivered to the customer for operational purpose is actually the second version after considering critical design and operational issues.
- In the first version, the team must have a special broad competence where they can quickly sense trouble spots in the design, model them, model alternatives.

4) Plan, control and monitor testing:

- The combination of manpower, computer time and management is called the test phase.
- This phase has greater risk in terms of cost and schedule
- The following are the important things in test phase:

1. Employ a team of test specialists who were not responsible for the original design;
2. Employ visual inspections to spot the obvious errors
3. Test every logic path
4. Employ the final checkout on the target computer

5) Involve the customer:

- It is important to involve the customer in a formal way so that he has committed himself at earlier points before final delivery.
- These include a "preliminary software review" following the preliminary program design step, a sequence of "critical software design reviews" during program design, and a "final software acceptance review“ after testing is performed.

Some software projects still follow the conventional software management process

The characteristics of conventional model are:

1. Protracted integration and late design breakage
2. Late risk resolution
3. Requirements-driven functional decomposition
4. Adversarial stakeholder relationships
5. Focus on documents and review meetings

1)Protracted integration and late design breakage

- Early success via paper designs and thorough briefings.
- Commitment to code late in the life cycle.
- Heavy budget and schedule pressure to get the system working.
- Late response of non optimal fixes, with no time for redesign.
- A very delicate, unmaintainable product delivered late

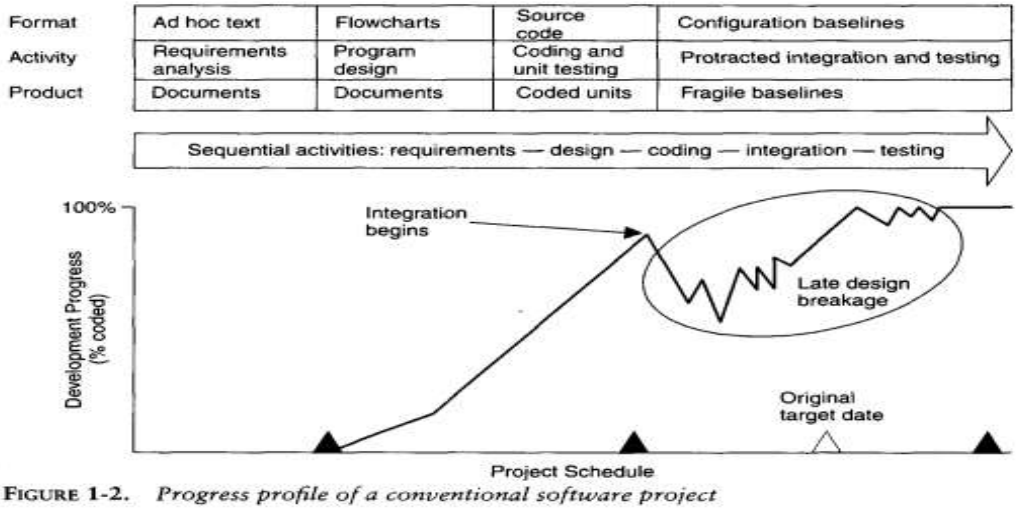


FIGURE 1-2. *Progress profile of a conventional software project*

- In the conventional model, the entire system was designed on paper, then implemented all at once, then integrated.
- Here we perform system testing at the end of the process to check the fundamental architecture is good or not

2)Late Risk Resolution:

- A serious issue associated with the waterfall lifecycle was the lack of early risk resolution

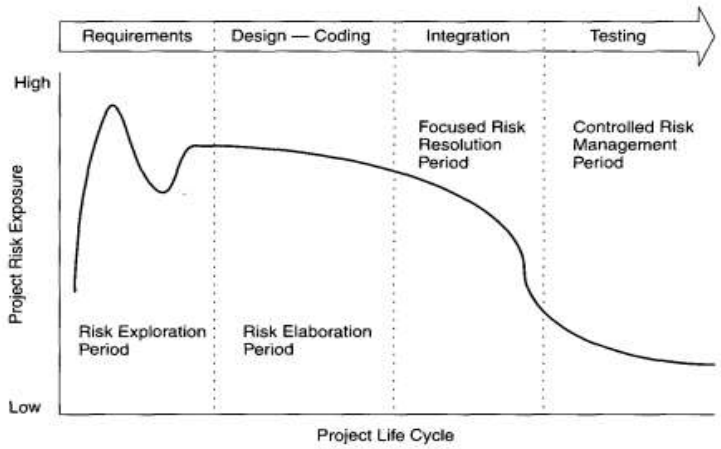


FIGURE 1-3. *Risk profile of a conventional software project across its life cycle*

- Early in the life cycle, as the requirements were being specified, the actual risk exposure was highly unpredictable.
- After a design concept available even on paper, risk exposure stabilized.
- In the next step, after the system was coded some of the individual component risks was resolved.
- When the integration begins the real system risks are touchable

3. Requirements-driven functional decomposition

- The software development process has been requirements driven.
- We should present precise requirements definition and then implement exactly those requirements.
- We should treat all the requirements are equally important.
- Requirement specification is an important and difficult job in the development process

4. Adversarial stake holder Relationship :

- The contractor prepared a draft contract-deliverable document and delivered it to the customer for approval.
- The customer was expected to provide comments
- The contractor incorporated these comments and submitted a final version for approval

5. Focus on documents and Review meetings :

- The conventional software development process focused on producing various documents that attempted to describe the software product.
- Here formal meetings were conducted to exchange specific documents.
- The developers produce tons of papers to describe the project progress to the customers rather than to reduce risk and produce quality software

Conventional software management process

Barry boehm's describes the objective characterization of the state of software development.

1) Finding and fixing a software problem after delivery costs 100 times more than finding and fixing the problem in early design phases

2) You can compress software development schedules 25% of normal, but no more

3) For every $1 you spend on development, you will spend $2 on maintenance

4) Software development and maintenance costs are primarily a function of the number of source lines of code.

5) Variations among people account for the biggest differences in software productivity

6) The overall ratio of software to hardware costs is still growing. In 1955 it was 15:85; in 1985, 85:15.

7) Only about 15% of software development effort is devoted to programming

8) Software systems and products typically cost 3 times as much per SLOC as individual software programs.

9) Walkthroughs catch 60% of the errors

10) 80% of the contribution comes from 20% of the contributors

- 80% of the engineering is consumed by 20% of the requirements
- 80% of the software cost is consumed by 20% of the components
- 80% of the errors are caused by 20% of the components
- 80% of software scrap and rework is caused by 20% of the errors
- 80% of the progress is made by 20% of the people

Software economics

Most software cost models can be described five parameters: size, process, personnel, environment and required quality.

1) The size of the end product which is typically measured in terms of the number of source lines or the number of function points developed to achieve the required functionality.

2) The process used to produce the end product, the ability of the process is to avoid non value adding activities

3) The capabilities of software engineering personnel, and particularly their experience with the computer science issues and the applications domain issues of the project

4) The environment, which is made up of the tools and techniques available to support efficient software development and to automate the process

5) The required quality of the product, including its features, performance, reliability, and adaptability

The following equation shows the relationship between size, process, personnel, environment, quality parameters and between the estimated cost

Effort = (Personnel) (Environment) (Quality) (Size process)

- Here one important feature of software economics is the relationship between effort and size exhibits is a diseconomy of scale.
- It indicates that more software we build it consumes more expensive per unit of the software
- So the per line cost of smaller applications is less than for the larger applications.
- The following diagram shows 3 generations software economics in the form of basic technology advancement in tools, components and processes

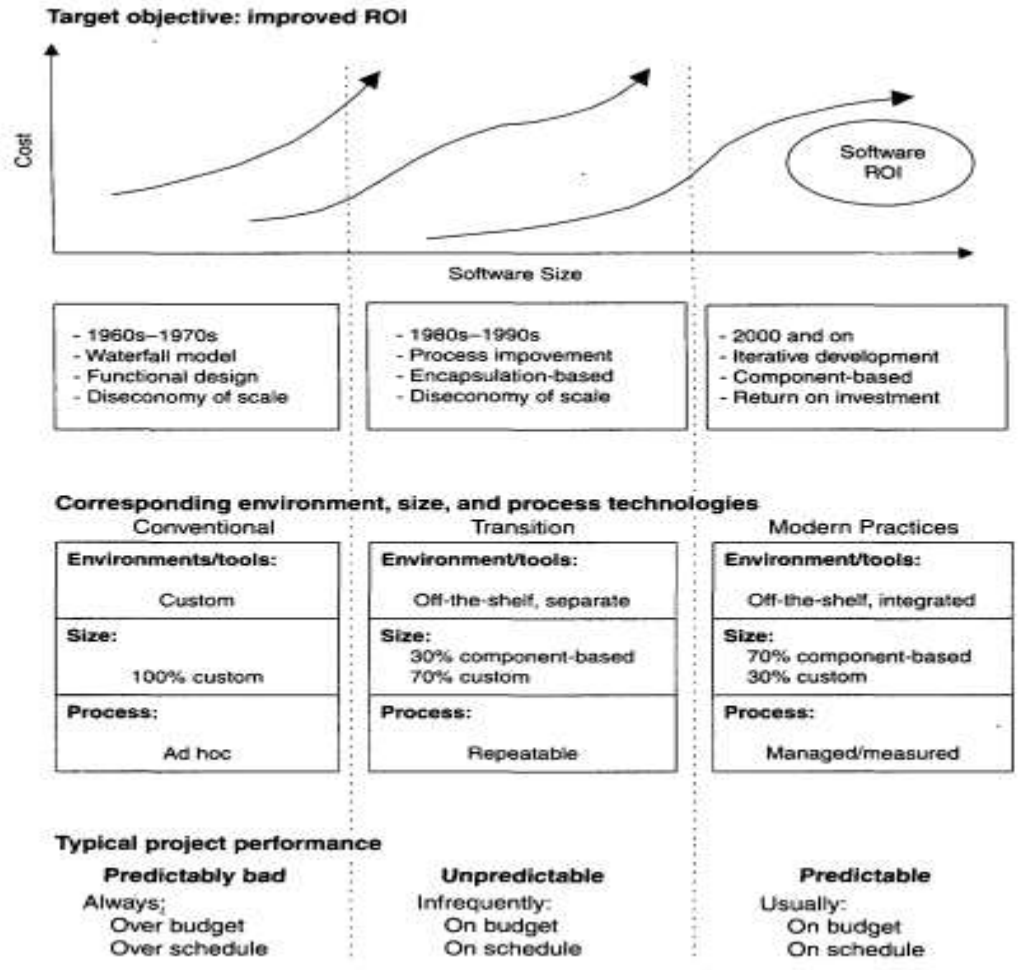


FIGURE 2-1. *Three generations of software economics leading to the target objective*

- The phases represents the life cycle of software business in the organization. so we can define the three generations of software development as follows:

1. **Conventional:**

- The software is developed in conventional manner between 1960 and 1970.
- Organizations used custom tools, custom processes, and virtually custom components built in primitive languages
- Here the performance of the project is highly predictable and cost, schedule and quality objectives were always under achieved

2. Transition:

- This is middle age of the software development which is between 1980 and 1990.
- At this time some software engineering principles are formed and organizations used repeatable and off the shelf tools
- During this phase custom components are built in higher level languages and Some of the components were available commercially such as operating system, database management system, networking, and graphical user interface.

3.Modern practices :

- This phase of software development process started in year 2000 and later.
- In this phase software development is treated as production.
- Here we use integrated automation environments, off the shelf components

Pragmatic software cost estimation

- One critical problem in software cost estimation is a lack of well-documented case studies of projects that used an iterative development approach.
- Software industry has inconsistently defined metrics or atomic units of measure the data.
- It is hard enough to collect a homogeneous set of project data within one organization with different processes, languages, domains, and so on

- There have been many debates among developers and vendors of software cost estimation models and tools.
    1. Which cost estimation model to use?
    2. Whether to measure software size in source lines of code or function points.
    3. What constitutes a good estimate?
- There are several popular cost estimation models (such as COCOMO, CHECKPOINT, ESTIMACS, Knowledge Plan, Price-S, ProQMS, SEER, SLIM, SOFTCOST, and SPQR/20)
- COCOMO is also one of the most open and well documented cost estimation models.
- SLOC worked well in applications that were predominantly custom built & SLOC measurement was easy to automate.
- Most real-world use of cost models is bottom-up rather than top-down

A good estimate has the following attributes:

- It is conceived and supported by the project manager, architecture team, development team, and test team accountable for performing the work.
- It is accepted by all stakeholders as ambitious but realizable.
- It is based on a well-defined software cost model with a credible basis.
- It is based on a database of relevant project experience that includes similar processes, similar technologies, similar environments, similar quality requirements, and similar people.
- It is defined in enough detail so that its key risk areas are understood and the probability of success is objectively assessed.

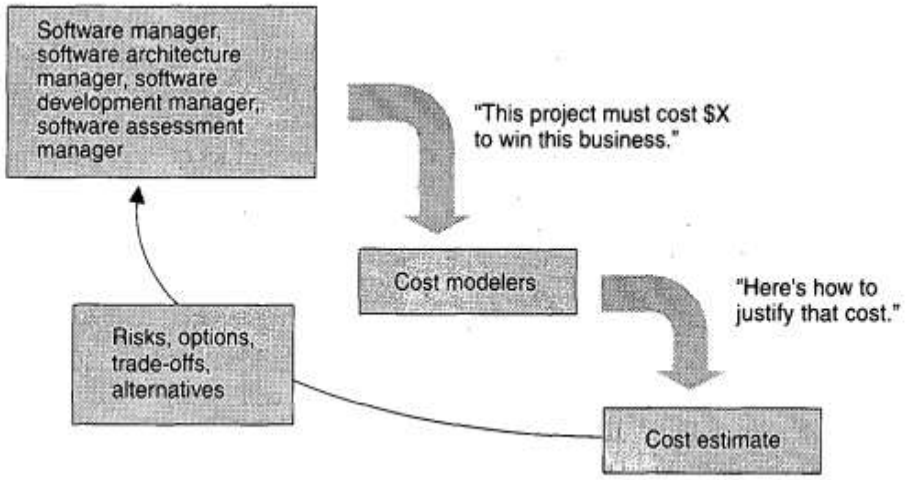


FIGURE 2-3. *The predominant cost estimation process*

- Extrapolating from a good estimate, an ideal estimate would be derived from a mature cost model with an experience base that reflects multiple similar projects done by the same team with the same mature processes and tools.