What is a Software?

Computer Programs and its documentation, s/w Products developed for a Particular Customer.

(or)

It is a collection of Programs and related data that Provides instructions for Computer what to do and how to do it.

what is software Engineering?

It is a Engginering decepline that Concerned with all aspects of software Production.

what is difference between Computer science and software Engineering?

Computer science is concerned with theory and fundamentals, software Engineering is concerned with the Practicalities of developing and delivery useful software

What is difference between system Engineering and Software Engineering?

System Engineering is Concerned with all aspects of Computer based system development, including hardware, Software, Process Engineering, software Engineering is Part of system Engineering.

what are the Framework activities of software Process?

1. Communication → Project initiation, Requirement galtering, Communication and collaboration with Customer.

Planning → estimation, schedule, tracking.
Modeling → Analysis, Design.
Construction → Code generation, testing.
Deployment → Delivery, Support, feedback.

what are the software Process models?

Waterfall Model, Incremental Model, RAD Model, Evolutinary Model, Prototype Model, Spiral Model etc.

Water-Fall Model:-

1. waterfall model is also called Classic life cycle & it is oldest model.
2. It suggests systematic, sequential approch
3. It begins with communication and ends with deployment.
4. The drawback is, Real Projects allows rarely, sequential flow.
5. This model is useful to serve where requirements are fixed.
6. when the changes occurs, cause Confusion as the Project team Proceeds, so that, it is difficult to Customer to state requirements in detail, finally delivery of Product with in time-span is not available.

what is difference between waterfall model and spiral model?

In waterfall model, requirements should be fixed, should not change, because it is sequential flow.

In spiral model, requirements can change, because it is Iterative flow.

UNIT-1

Conventional Software Management:

→ The best thing about software is its "flexibility". It can be programmed to do "almost anything".

→ The worst thing about software is also "flexibility": "The almost anything" characteristic has made it difficult to plan, monitor and control software development.

→ In the mid-1990's atleast 3 important analysis of the state of the software engineering industry were performed.

→ The results were presented in :

1. Patterns of software system failure and success [Jones, 1996].
2. Chaos [Standish Group, 1995]
3. Report of the Defense Science Board Task Force on Acquiring Defense Software Commercially [Defense Science Board, 1994]

All three analysis reached the same general conclusion: They can be summerized as follows:

1. Software development is still highly unpredictable only about 10% of Sw Projects are delivered successfully within budget and schedule estimates.
2. Management discipline is more of a discriminator in success or failure than are technology advances.
3. The level of software scrap and rework is indicative Process.

→ The 3 analysis Provide the magnitude of software Problem and the current norms for Conventional software Management Performance

→ The software management Process framework, that most comp. Conventional software Projects have used, while this framework, known as the waterfall model.

1.1 The Waterfall Model :-

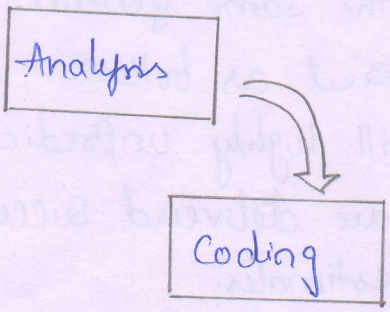
Most Software engineering tests Present the waterfall model as the source of the "Conventional" S/w Process. This section examines and critiques the waterfall model theory, then looks at how most of the industry has Practiced the Conventional S/w Process.

In Theory :-

→ In 1970's, my father (Walker Royce's father) winston Royce Presented a Paper titled "Managing the development of large scale S/w S/m" at IEEE WESCON.

→ This Paper made 3 Primary Points:

1. There are 2 essential steps common to the development of Computer Programs: analysis and coding



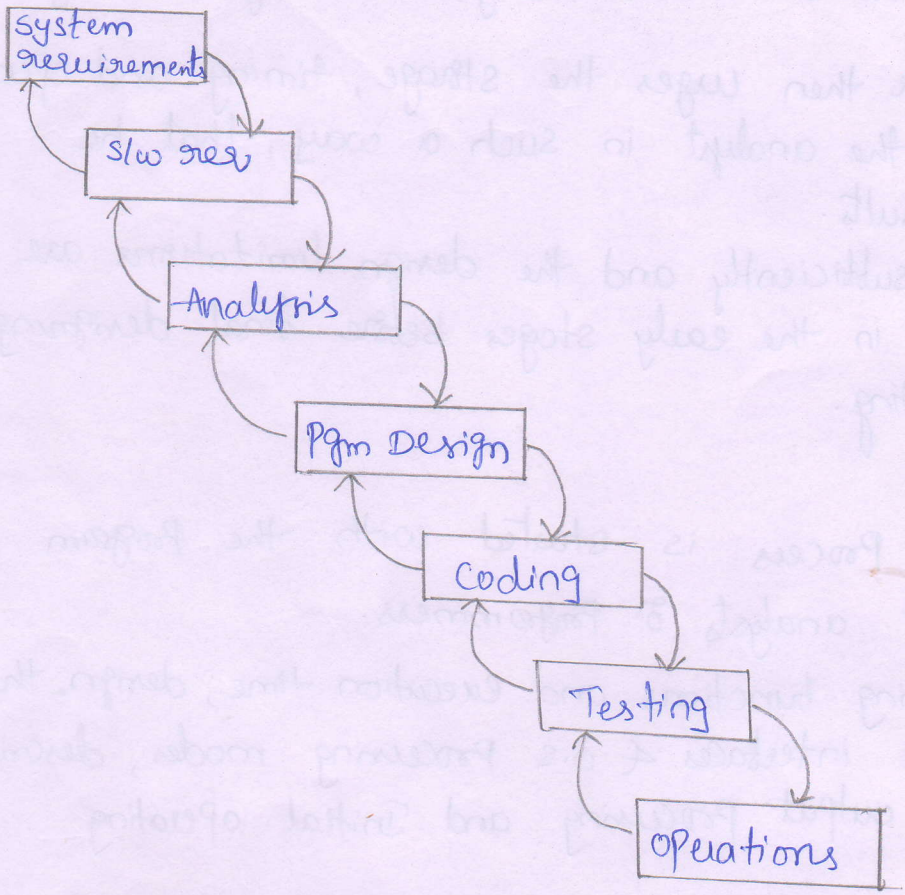
→ Analysis and coding both involve Creative work that directly Contributes to the usefulness of the end Products

2. The order to manage and control the software development it includes system requirements, software requirements, analysis, Program design, coding, Testing and operations. There are supplement to the analysis and

Coding steps. Figure shows:

The resulting Project Profile and the basic steps in developing a large-scale Program.

Fig: The large-scale system approch



3.

Five Necessary improvements for this approch to work

1. Complete Program design before analysis and coding design.
2. Maintain current and complete documentation
3. Do the job twice, if possible
4. Plan, Control and monitor testing
5. Involve the Customer.

→ Five improvements to the basic waterfall Process that would elimitate most of the development risks.

1. Program Design Comes First :-

→ The first step is to insert an initial design phase between the requirements phase and analysis phase.

→ Hence, by using this technique, S/w failure will not occur due to the continuous change in storage, timing and data.

→ The designer then urges the storage, timing and operational limitations on the analyst in such a way, that he notices the results.

→ Resources insufficiently and the design limitations are then identified in the early stages before final designing coding and testing.

Design steps:-

(i) The design process is started with the program designed, but not analysts & programmers.

(ii) Allocate processing functions and execution time, design the database, define interfaces & O.S processing modes, describe the input and output processing and Initial operating procedures.

(iii) Design an overview document in such away that, it is understandable, correct and informative.

2. Document the Design :-

The amount of documentation associated with the S/w Pgms is very large because of the following reasons,

(i) The S/w designer has to communicate with the interfacing designers, managers and customers. Hence, requiring large amount of documentation.

(ii) In the earlier phases of the S/w development, the documentation is the design.

(iii) The real financial value of the documention is decided in such a way that, it will support future modifications made by an isolated test team, maintenance team and operations personnel.

3. DO the Job twice, if Possible:-

The Computer Program must be developed twice and the second version, which takes into account all the critical design operations, must be finally delivered to the Customer for operational deployment. The first version of the Computer Program involves a special broad Competence team, responsible for notifying the troubles in design, followed by their modeling and finally generating an error-free Program.

4. Plan, Control and Monitor Testing:-

→ The test phase is the biggest user of the Project resources, such as manpower, Computer time and Management assessments.

→ It has the greatest risk in terms of Cost and schedule and develops at the most recent Point in the schedule, when backup alternatives atleast available.

→ Thus, most of Problems need to be solved before the test phase, as it has to Perform some other important operations.

(i) Hire a team of test specialists, who are not involved in the original design.

(ii) Apply visual inspections to discover the obvious errors, such as skipping to wrong addresses, dropping of minus signs etc,

(iii) Conduct a test for every logic path.

(iv) Employ the final checkout on the target Computer.

5. Involve the Customer:-

→ The Customer must be involved in a formal way, so that, he has devoted himself at the initial stages before final (~~initial design step~~ ~~botto~~) delivery.

→ The Customer Perception, assessment and Commitment Can Strengthen the development effort.

→ Hence, an initial design step followed by "Preliminary S/w review" "Critical s/w design reviews", during design and a "final s/w acceptance review" after testing is Performed.

IN Practice:-

→ The characteristics of the conventinal Process as it has typically been applied, which is not necessarily as it was intended.

→ Projects destined for trouble frequently exibit the following symptoms:

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

unpleasant (or) worse than you hoped

Protracted integration and late design breakage:-

For a typical development Project that used a waterfall model management Process, fig below illustrates development Progress versus time.

Progress is defined as Percent Coded ie, demonstrable in its target form. The following sequence was common:

1. Early success via Paper designs and through briefings.

is a meeting at which info or instructions are given to people, especially before they do something.

2. Commitment to code late in the lifecycle.
3. Integration nightmares due to unforeseen implementation issues and interface ambiguities.
4. Heavy budget and schedule pressure to get the system working.
5. Late shoe-horning of nonoptimal fixes, with no times for redesign.
6. A very fragile, unmaintainable product delivered late.

Format	Adhoc text	Flow charts	Source code	Configuration baselines
Activity	Requirements analysis	Program design	Coding and unit testing	Protracted integration and testing.
Product	Documents	Documents	Coded units	Fragile baselines

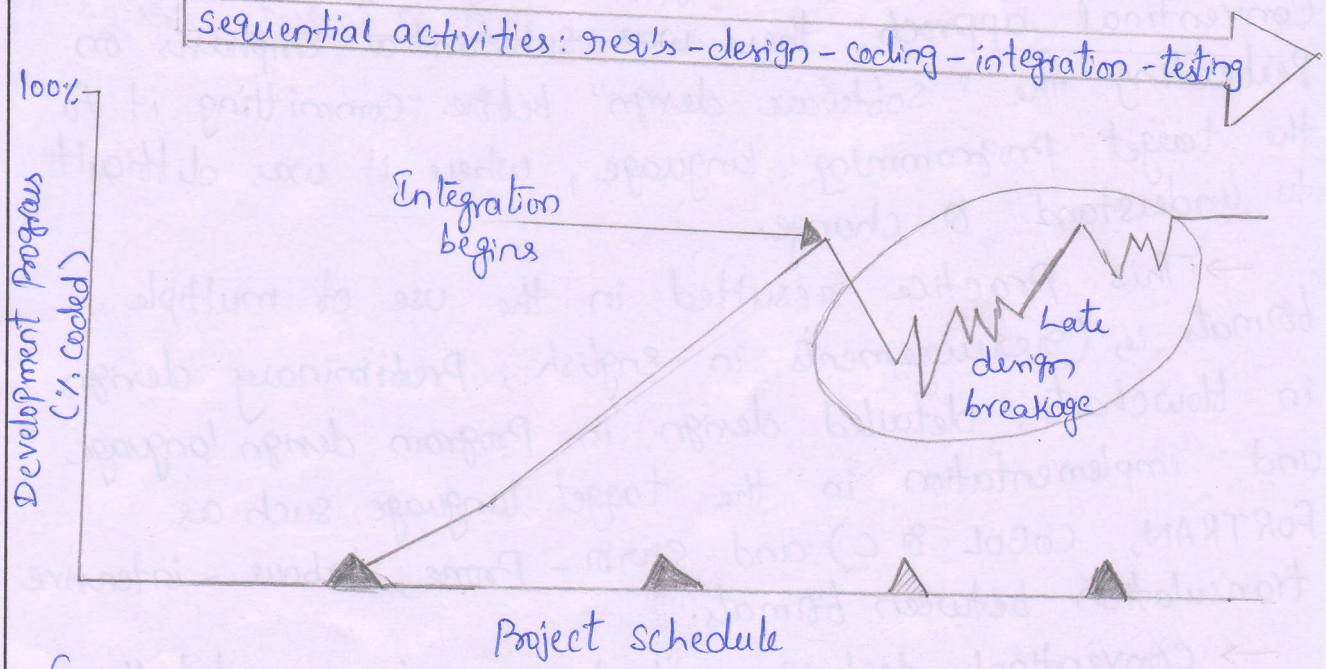


Fig: Progress Profile of a Conventional Software Project

Table: Expenditures by activity for a Conventional software Project.

Activity	COST
Management	5%
Requirements	5%
Design	10%
Code and unit testing	30%
Integration and test	40%
Deployment	5%
Environment	5%
Total	100%

→ The immature languages and technologies in the Conventional approch, there was substantial (large in amount, special importance) Emphasis on Perfecting the "software design" before committing it to the target programming language, where it was difficult to understand & change.

→ This Practice resulted in the use of multiple formats ie (requirements in English, Preliminary design in flowcharts, detailed design in Program design language and implementation in the target language such as FORTRAN, COBOL & C) and error-Prone, labour-intensive transulation between formats.

→ Conventional techniques that imposed a waterfall model on the design Process inevitably resulted in late integration and Performance showstoppers.

→ In Conventional model, the entire system was designed on Paper, then implemented all at once, then integrated.

→ only at the end of this process was it possible to perform system testing to verify that are fundamental architecture was sound.

→ The conventional process was that testing activities consumed 40% or more of life-cycle resources.

Late Risk Resolution:-

→ A serious issue associated with the waterfall life cycle was the lack of early risk resolution.

→ Fig: below shows a typical risk profile for conventional waterfall model projects. It includes four distinct periods of risk exposure, where risk is desig. defined as the probability of missing a cost, schedule, feature or quality goal.

→ Early in the lifecycle, as the req.'s were being specified, the actual risk exposure was highly unpredictable.

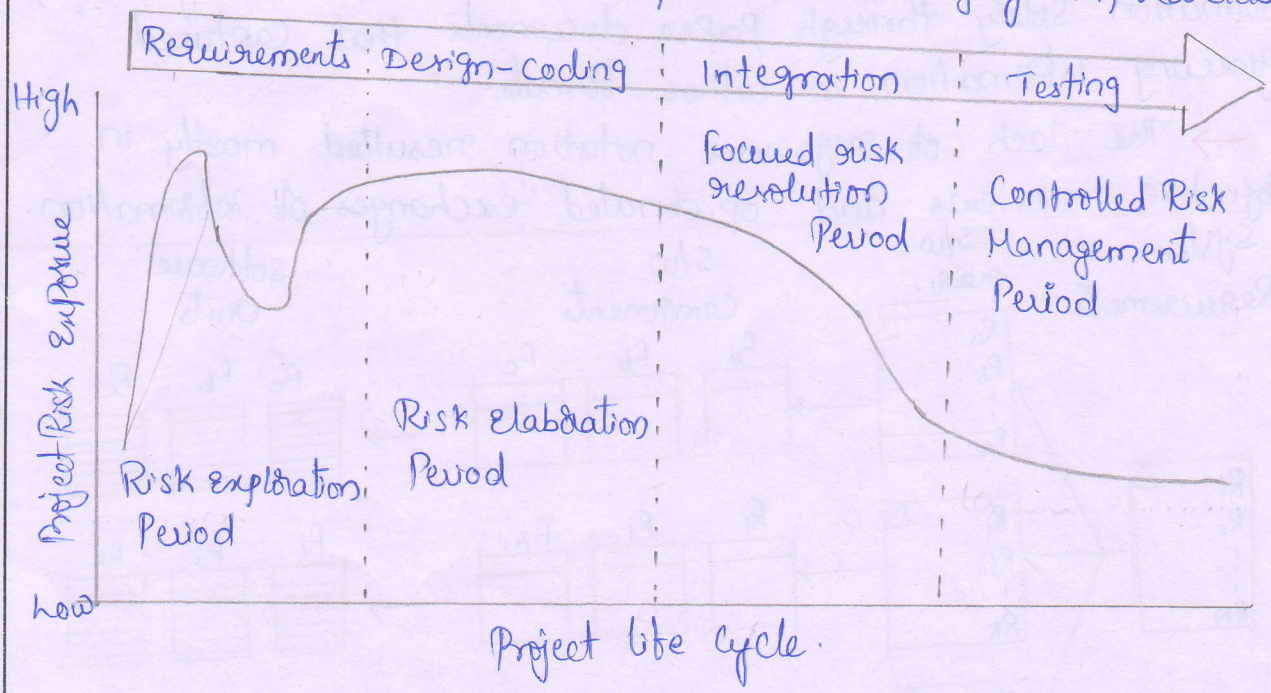


Fig: Risk Profile of a conventional software project across its lifecycle

→ After a design concept was available to balance the understanding of the requirements, even if it was just on paper, the risk exposure stabilized.

→ As the system was coded, some of the individual component risks got resolved. Then integration began, and the real system-level qualities and risks started issues becoming tangible. clear, earlyseen

→ The late risk resolution is expensive and the protracted integration phase of the project resolves the risks without affecting the quality of the end product. Thus the design integrity and maintainability are not conserved.

protect it from changes & cure & harm

Adversarial Stakeholder Relationships:-

→ The Conventional Process tended to result in adversarial stakeholder relationships, in large part because of the difficulties of requirements specification and the exchange of information solely through paper documents that captured engineering information in adhoc formats.

→ The lack of rigorous notation resulted mostly in subjective reviews and opinionated exchanges of information.

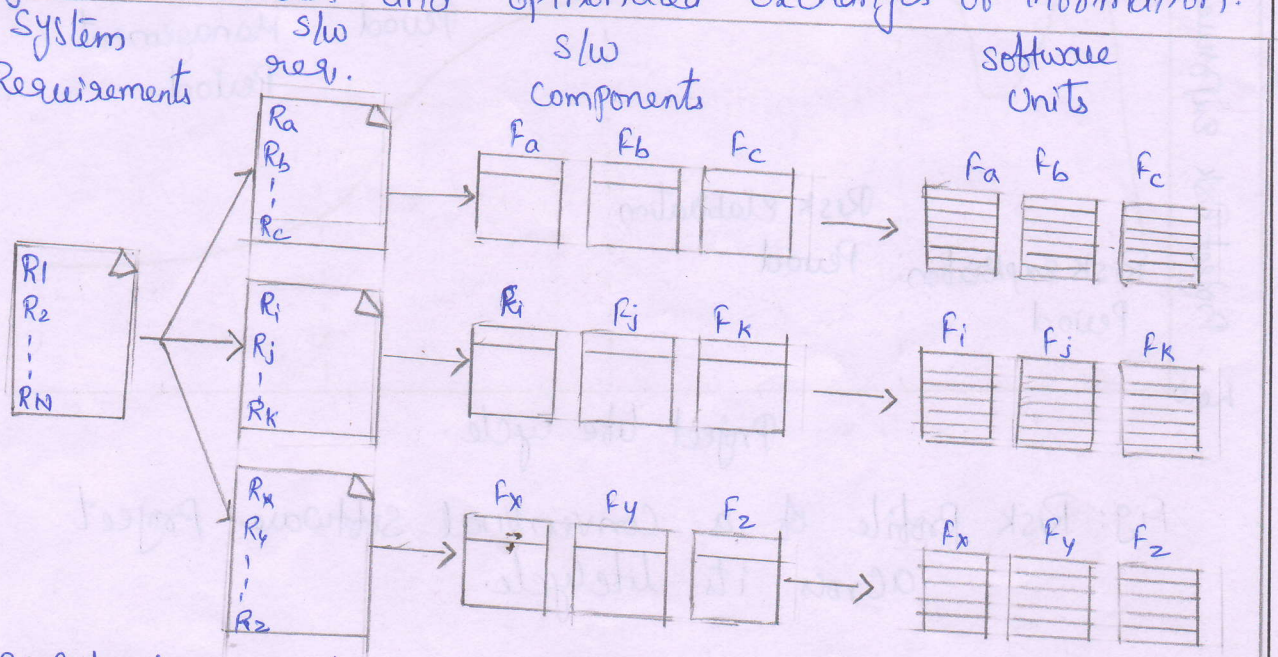


Fig: Suboptimal software component organization resulting from a (Constractual software) Req's driven approch

The following sequence of events was typical for most contractual software efforts:

1. The Contractor prepared a draft contract - deliverable document that captured an intermediate artifact and delivered it to the Customer for approval.
2. The Customer was expected to provide Comments
3. The Contractor incorporated these Comments and submitted a final version for approval.

→ This one-shot review Process encouraged high levels of Sensitivity on the Part of Customers and Contractors.

→ This approch also resulted in Customer - Contractor r/s degenerating into mutual distrust, making it difficult to acheive a balance amount requirements, design Schedule & Cost.

Requirements - Driven Functinal Decomposition:-

→ The S/w development Process has been req's driven i.e. initially gives precise defination for the requirements and then provides implementation for them.

→ The req's are specified Completely & clearly before any other activities in the S/w development Process. It immaturely treats all the requirements.

→ Req's specification is an important and difficult job in the development Process.

→ The Concept of equally testing the req's depletes & (stock means to reduce it.) takes significant engineering hours actually associated with traceability, testability, logistics support etc,

→ The basic assumption of the waterfall Process is that, req's are specified in a functinal manner, a/c to which the S/w can be divided into functions followed by the req's allocation to the resulting components.

Focus on Documents and Review Meetings :-

→ The conventional process focused on producing various documents that attempted to described the s/w product, with insufficient focus on producing tangible increments of the products themselves.

→ Major milestones were usually implemented as ceremonious meetings defined solely in terms of specific documents.

→ Contractors were driven to produce literally tons of paper to meet milestones and demostrate progress to stake holders, rather than spend their energy on tasks that would reduce risk and produce quality s/w.

→ Typically, Presenters and the audience reviewed the simple things that they understood rather than the complex & important issues.

→ Most design reviews, ∴ resulted in low engging value & high cost in terms of the effort & schedule involved in their preparation & conduct.

Table : Results of conventional s/w project design reviews

Apparent Results	Real Results
Big briefing to a diverse audience	only a small % of the audience understands the s/w. Breifings & documents expose few of the important assests & risks of complex s/w s/ms.
A design that appears to be compliant.	There is no tangable evidence of compliance. Compliance with ambigous reqts is of little value.
Coverage of requirements (typically hundreads)	Few (tens) are design divers. Dealing with all reqts dilutes the focus on the critical divers.
A design considered "innocent until proven guilty"	The design is always guilty. Design flaws are exposed late in life cycle.

Conventional Software Management Performance:-

→ Barry Boehm's one-page "Industrial software Metrics Top 10 List" is a good, objective characterization of the state of s/w development.

(In the following paragraphs, quotations from Boehm's top 10 list are presented in italics, followed by my comments)

1. Finding and fixing a s/w 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 nominal, 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 s/w to h/w costs is still growing. In 1955 it was 15:85; in 1985, 85:15.
7. Only about 15% of s/w development effort is devoted to programming.
8. Software systems and products typically cost 3 items as much per SLOC as individual s/w pgms. S/w-s/m's products cost 9 times as much.
9. Walkthrough's 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 s/w cost is consumed by 20% of the components.

→ 80% of the errors are caused by 20% of the components.

→ 80% of s/w scrap and rework is caused by 20% of the errors.

→ 80% of the resources are consumed by 20% of the components.

→ 80% of the engineering is accomplished by 20% of the tools.

→ 80% of the progress is made by 20% of the people.

These relationships provide some good benchmarks for evaluating process improvements and technology improvements.

Evolution of Software Economics:-

Software economics:-

Most software cost models can be abstracted into a function of 5 basic parameters: size, process, personnel, environment and required quality.

1. The "size" of the end product, which is typically quantified in terms of the number of source instructions or the number of function points required to develop the required functionality.
2. The "process" used to produce the end product, in particular the ability of the process to avoid non-value-adding activities.
3. The capabilities of s/w engg "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 & techniques available experience with the computuscience issues and to automate the process
5. The required quality of the product, including its features

Performance, reliability and adaptability.

The relationship among these parameters and the estimated cost can be written as follows:

$$\text{Effort} = (\text{Personnel}) / (\text{Environment}) (\text{Quality}) (\text{Size}^{\text{Process}})$$

→ several parametric models have been developed to estimated software costs; all of them can be generally abstracted into this form.

→ One important aspect of software economics is that the the relationship between effort and size exhibits a diseconomy of scale.

→ The diseconomy of scale of s/w development is a result of the process exponent being greaterthan 1.0.

→ Contrary to most manufacturing processes, the more software you build, the more software you bilud, the more expensive it is per unit item.

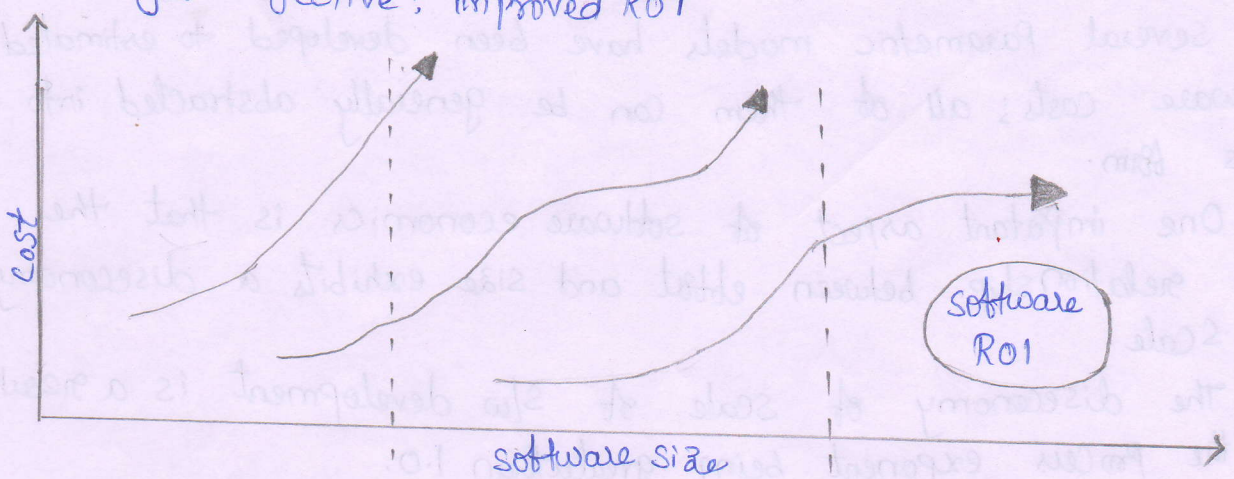
Eg:- For a given application, a 10,000 lines s/w solution will cost less per line than a 100,000 s/w solution. How much less?

- Assume that a 100,000-line system requires 900 staff-months for development, or about 111 lines per staff-month, or 1.37 hours per line.
- If this same s/m were only 10,000 lines, and all other parameters were held constant, this project would be estimated at 62 staff-months or about 175 lines per staff-month, or 0.87 hour per line.
- The per-line cost for the smaller application is much less than for the larger application.

→ Fig: below shows 3 generations of basic technology advancement in tools, components, and processes. The required levels of quality and personnel are assumed to the constant

→ The ordinate of the graph refers to software unit costs realized by an organization.

Fig: Three generations of software economics leading to the target objective



- 1960s - 1970s - waterfall model - Functional design - Diseconomy of scale	- 1980s - 1990s - Process improvement - Encapsulation-based - Diseconomy of scale	- 2000 and on - interactive development - Component-based - Return on investment

Corresponding Environment, Size, and Process technologies

Conventional	Transition	Modern Practices
Environments/tools: Custom	Environments/Tools: off-the-shelf, separate	Environments/Tools: off-the-shelf, integrated
Size: 100% Custom	Size: 30% Component-based 70% Custom	Size: 70% Component-based 30% Custom
Process: Adhoc	Process: Repeatable	Process: Managed/measured

Typical Project Performance

Predictably bad	Unpredictable	Predictable
Always: over budget over schedule	Intermediately: on budget on schedule	Usually: on budget on schedule

The abscissa represents life cycle of the s/w business engaged in by the organization. The 3 generations of s/w development are defined as follows:

1. Conventional :- 1960's & 1970s, Craftmanship. Organizations used custom tools, custom processes and virtually all custom components built in primitive languages. Project performance was highly predictable in that cost, schedule, and quality objectives were almost always underachieved.

2. Trasition: 1980s and 1990s, s/w Engineering. Organizations used more-repeatable processes and off-the-shelf tools, and mostly (>70%) custom components built in higher level languages.

→ Some of components (<30%) were available as commercial products, including the O.S, DBMS, networking & graphical user interface (GUI).

→ During the 1980s, some organization began achieving economics of scale, but with the growth in application complexity, the existing languages, techniques and technologies were not just enough to substain the desired business performance.

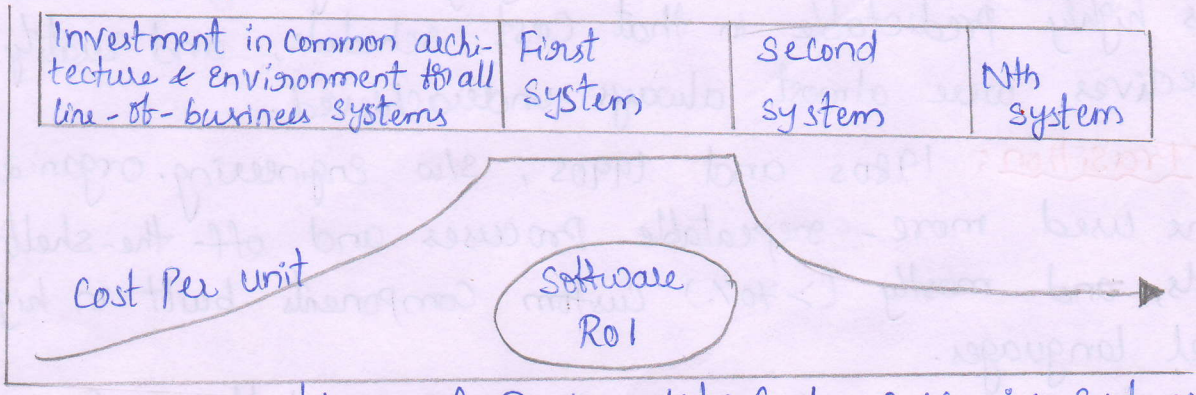
3. Modern Practices:- 2000 and latu, s/w production. It is rooted in the use of managed and measured process, integrated automation environments and mostly (70%) off-the-shelf components.

→ As few as 30% of components need to be custom built. With advances in s/w technology and integrated production environments, these component-based system can be produced very rapidly.

→ Technologies for environment automation, size reduction and process improvements are not independent of one another. In each new era, the key is complementary growth in all technologies.

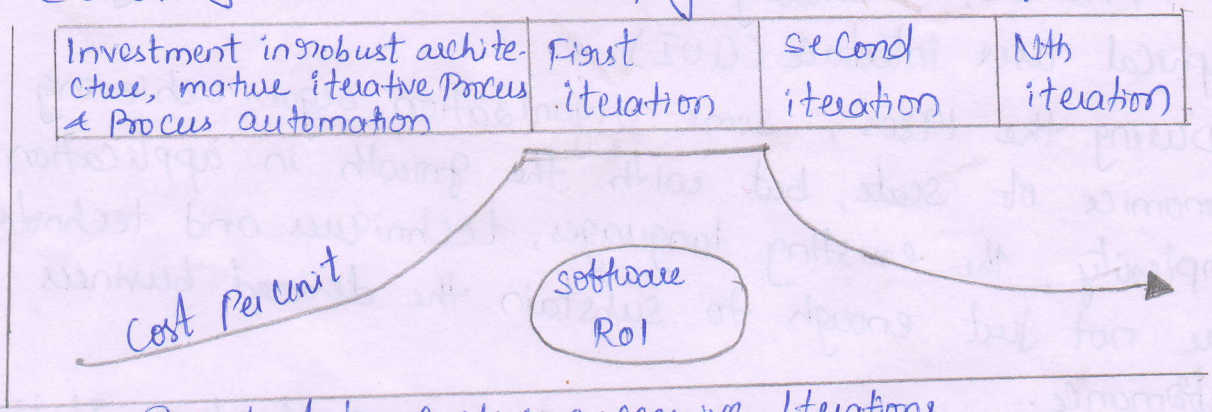
Fig: Return on investment in different domains

Achieving ROI across a line of business



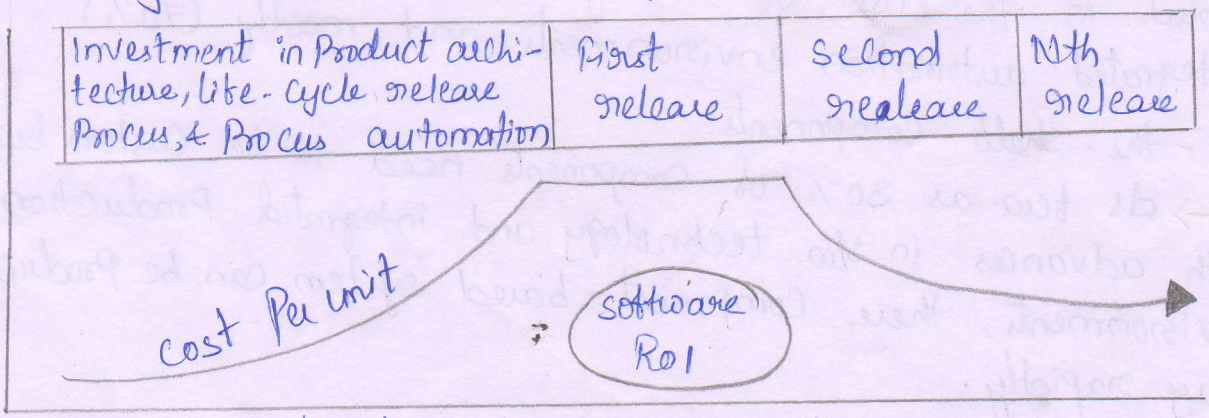
Line-of-Business life cycle: successive systems

Achieving ROI across a project with multiple iterations



Project life cycle: successive Iterations

Achieving ROI across a life cycle of Product releases



Project life cycle: successive Releases

→ Organizations are achieving better economies of scale in successive technology eras - with very large projects, long-lived products and lines of business comprising multiple similar projects

→ Figure above provides an overview of how a return on investment (ROI) profile can be achieved in subsequent efforts across life cycles of various domains.

Dealing with something is based on practical considerations

Pragmatic Software Cost estimation:-

One critical problem in s/w cost estimation is a lack of well-documented case studies of projects that used an iterative development approch.

Although cost model vendors claim that their tools are suitable for estimating iterative development projects, few are based on empirical project databases with modern iterative development success stories

Furthermore, because the software industry has inconsistently defined metrices or atomic units of measure, the data from actual projects are highly suspect in terms of consistency and comparability.

It is hard enough to collect a homogeneous set of project data within one oganizations with different processes, langua ges, domains and so on.

Eg- The fundamental unit of size can be, and is, counted differently across the industry. It is suprising that modern language standards don't make a simple definition of a source line reportable by the compiler.

The exact definition of a function point or a SLOC is not very important, that everyone just as the exact length of a foot or a meter is equally arbitrary. It is simply important that everyone uses the same definition.

There have been many long-standing debates among developers and Vendors of S/w Cost estimation models & tools. Three topics of these debates are of particular intrest here:

1. which Cost estimation model to use.
2. whether to measure S/w size in SLOC or FP
3. what constitutes a good estimate.

→ About 50 vendors of s/w cost estimation tools, data & services compete within the s/w industry.

→ There are several popular cost estimation models (such as COCOMO, CHECKPOINT, ESTIMACS, knowledge plan, SLIM), as well as numerous organization-spetic models. Because my firsthand experience with these models has been centered on COCOMO and its successors, Ada COCOMO & COCOMO II, it is the basis of many of my S/w economics arguments and Perspectives.

→ COCOMO is also one of the most open & well-documented Cost estimation models.

→ The measurement of S/w size has been the subject of much rhetoric

→ There are basically 2 objective Points of View: Source lines of Code and function Points.

→ Both Perspectives have Proven to be more valuable than a third, which is the subjective or adhoc Point of view Practiced by many immature organizations that use no systematic measurement of size.

→ Many S/w experts have argued that SLOC is a lousy measures of size.

However, when a code segment is described as a 4000-source-line Program, most People feel comfortable with

its general "mass". If the description were 80 function points, 6 classes, 5 use cases, 4 object points, 6 files, 2 subsystems, 1 component, or 6,000 bytes, most people, including s/w experts, would ask further questions to gain an understanding of the subject code. So SLOC is one measure that still has some value.

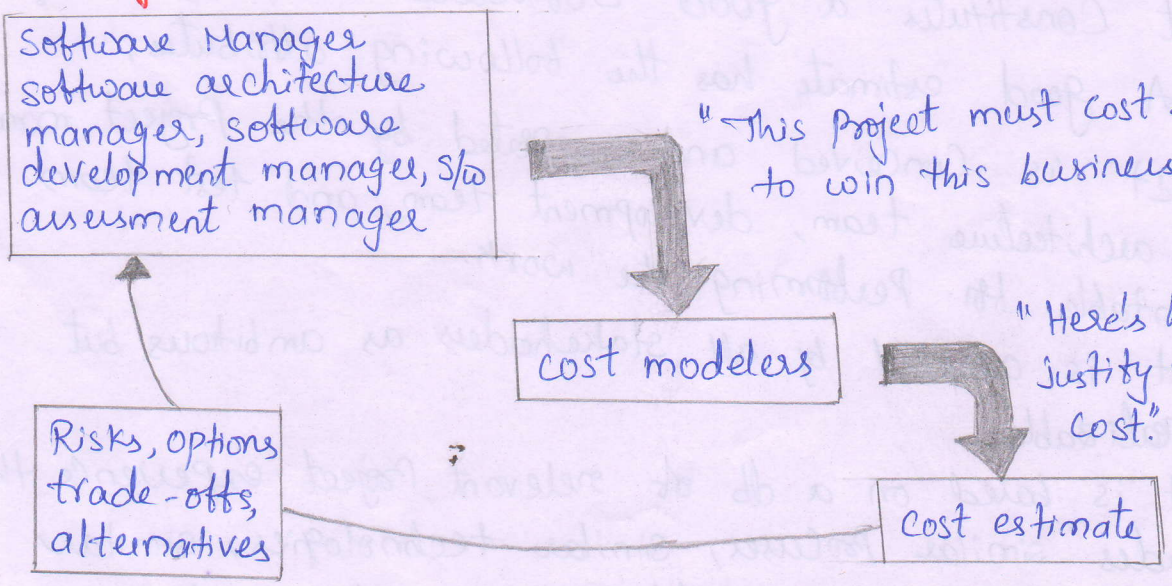
→ SLOC worked well in applications that were predominantly custom-built & because SLOC measurement was easy to automate & instrument.

→ Today, language advances & the use of components, automatic source code generation, and other orientation have made SLOC a much more ambiguous measure.

→ The general accuracy of conventional cost models has been described as "within 20% of actuals, 70% of the time".

→ This level of unpredictability in the conventional s/w development process should be truly frightening to every investor, especially in light of the fact that few projects miss their estimate by doing better than expected.

Fig: The Predominant Cost Estimation Process



→ Most real-world use of cost models is bottom-up rather than top-down.

→ Figure above states that the Predominant Practice: The S/w Project manager defines the target cost of the S/w, then manipulates the Parameters and sizing until the target cost can be justified.

→ The rationale for the target cost may be to win a Proposal, to solicit customer funding, to attain internal Corporate funding, or to achieve some other goal.

→ The Process described in figure above is not all bad. In fact, it is absolutely necessary to analyze the cost risks, and understand the sensitives and trade-offs objectively.

→ It forces the S/w Project manager to examine the risks associated with achieving the target costs & to discuss this information with other stakeholders.

→ The result is usually various Perturbations in the Plans, designs, Process, or scope being Proposed.

→ This Process Provides a good vehicle for a basis of estimate & an overall cost analysis.

"What Constitutes a good software cost Estimate?"

→ A good estimate has the following attributes:

1. It is Conceived and supported by the Project manager, architecture team, development team, and test team accountable for Performing the work.
2. It is accepted by all stakeholders as ambitious but realizable.
3. It is based on a db of relevant Project experience that includes similar Processes, similar technologies, similar environments, similar quality requirements & similar people.

4. It is based on a well-defined s/w cost model with a credible basis.

5. It is defined in enough detail so that its key risk areas are understood & the probability of success is objectively assessed.

→ 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.