The Mythical Man-Month: Essays on Software Engineering

by

Frederick P. Brooks, Jr.

- Brooks => Manager for Architecture & Operating System (OS360) development for IBM360 & was also involved in design and implementation of the IBM Stretch.
- OS360 is an important product since it introduced several innovative ideas such as:
- Device independent I/O
- External Library Management
- This book is for professional managers of software development projects.
- Brooks believes large program development management suffers from problems related to division of labor.
- Brooks says critical problem is preservation of conceptual integrity of the product itself => he will propose methods for preserving this unity in Chpt's 2-7.

- **Chpt 1: The Tar Pit**
- **Large system programming is like a tar pit:**
- **The more you struggle, the more you sink in it.**
- **No one thing seems to be the difficulty, but as a whole, the programmers get trapped in it and the project dies (e.g., we can take one paw or hand out of the tar pit, but not our whole body).**
- **A programming system product must be:**
  - **1) General (for input, as well as algorithms).**
  - **2) Thoroughly tested so it can be depended upon. This increases costs be as much as 3 times over untested products.**
  - **3) Documented so it can be used, fixed, and extended.**
  - **4) Finally, a programming product becomes a component of a software system (it works with other programs and can be called or interrupted) => I/O interfaces become very important => a programming product must be tested in all possible combinations with other system components. This is very difficult => A programming system product as a system component costs at least 3 times as much as the same program as a programming product.**
  - **5) Programming product must be within allotted budget.**
- **A programming system product is 9 times higher in cost than a stand-alone product (3 for program testing x 3 for system testing).**

- **Chpt 2: The Mythical Man-Month**
- **What one factor has caused the demise of software projects?**
- **Poor scheduling and timing estimations!**
- **Problems with scheduling and timing estimations in a software project:**
  - **1) Estimation of the completion time for the software projects is typically very difficult to do.**
  - **2) In estimating the completion time of software projects we often assume that the effort and progress are the same, but that's not true. Especially we tend to interchange MAN and MONTH.**
  - **3) Scheduled progress is poorly monitored.**
  - **4) When software projects get behind the scheduled time, the typical solution is to add more MAN-MONTH or manpower to the software project.**
- **In software projects we always assume that everything will go well!**
- **In a non-software engineering project we often run into unforeseen problems that are intractable, and therefore the managers tend to over-estimate in timing requirements to compensate for them.**
- **In software engineering, however, we always think of programming as something which is tractable and which is documented. However, this assumption is not correct because we always have bugs in software; and although bugs are tractable, they are very difficult to detect. That will make the problem almost intractable.**

- **The second problem with estimating completion time of software projects is with the unit of time used to estimate the completion of the project (i.e., MAN-MONTH).**
- **There is a fallacy involved with this unit: It is true that the cost of the project will increase proportional to the MAN-MONTH used for that project. However, the progress is not going to be achieved proportional to the number of MAN-MONTHS used for a project.**
- **Therefore, it is a false assumption that by adding more MAN-MONTHS you can complete the project earlier.**
- **MAN and MONTH are in fact interchangeable, only if the project consists of several independent efforts (INDEPENDENT PARTITIONED TASKS) and a case where team members do not have to communicate with each other.**

e.g.:	**Men for**	**Months**	**Man-Month**
	5	**4**	**20**
	X2	**/2**	
	10	**2**	**20**

- **In fact, one of the problems with software engineering projects is that as we add more MAN-MONTHS, the amount of communication among the men is going to increase rapidly.**

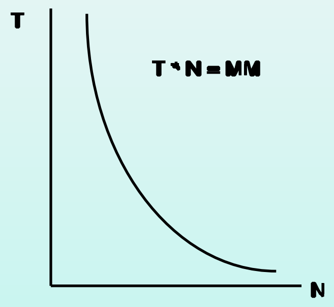
- Therefore, by adding more MAN-MONTHS, we are, in fact, increasing the completion time of the software project because the communication is going to affect the completion time of the project.
- Thus, the assumption that MAN and MONTH are interchangeable is not correct.
- Another very important factor that affects our estimation of completion time of software projects is the system testing and debugging process.
- Brooks' rule of thumb for estimating the completion time of software projects:
  - 1/3 time for planning;
  - 1/6 time for coding;
  - 1/4 time for component tests;
  - 1/4 time for system test with all components in hand.
- Note that the testing and debugging takes about 50% of the time, while coding, which people often worry about, only takes about 1/6 of the time.
- Actually, coding is the easiest phase to accurately estimate!
- Underestimating the time it takes for system testing can become very dangerous because it comes at the end of the project:
  - project is already fully funded and has the most manpower allocated to it,

  - the customer is ready to receive the product,
  - any delays at that point is going to increase the cost of the system dramatically,
  - and it is going to reduce the confidence of the customer.
- One other problem with the scheduling of software projects is that software managers often estimate the completion time of the projects according to the desired date of the customers.
- Instead of making the schedules realistic they try to match what the customer wants and therefore get into trouble later.

- **Brooks' Law:**
  - **Adding manpower to a late software project makes it later.**

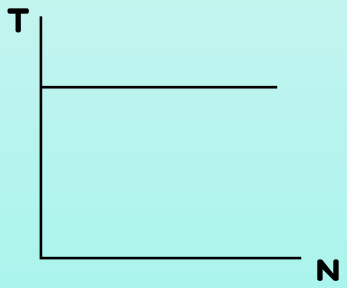
- The number of months of a project depends upon its sequential constraints. The maximum number of men depends upon the number of independent subtasks. From these two quantities one can derive schedules using different number of men and months.

PARTIONABLE TASK



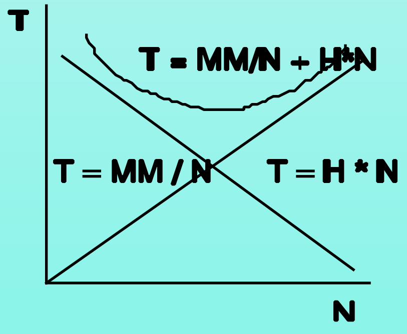
As persons are added each is given an equal share of the task to perform.
No communications requirement.
Ex : Painting a wall with your
own paint and tools.

UNPARTIONABLE TASK



Each new added person takes on the whole task.
Ex : Having a baby,
Learning something.
Each person involved must go through the entire thing.

UNPARTIONABLE TASK WITH COMMUNICATION



Assumes time for communocation is a linear function of N.
A bit more realistic !
Or is it ?

- Let *MMc* be the average effort expended by a pair of persons, working on the project, in communicating with each other - determined to be based on the project not *N*. Then:

$$\sum MMc = \frac{N^2 - N}{2} \times MMc \qquad = \qquad Pairs \times MMc$$

  – Assumes no original level isolations. But <u>complete communication</u> among team members.

- MMn = Effort required without communication
- MM = Total effort including communication:

$$MM = MMn + N\left(\frac{N-1}{2}\right)MMc \quad = N \times T$$

$$T = \frac{MMn}{N} + \frac{N-1}{2} MMc$$

$$\approx \frac{MMn}{N} + \frac{MMc}{2} \times N \qquad (if\ \ N >> 1), \quad let\ H = \frac{MMc}{2}$$

$$T \approx \frac{MMn}{N} + H \times N$$

- **Note : This is appropriate for task force , committee and democratic organizations. Not so for others.**

- **FINDING THE OPTIMUM TASK STAFFING**
  - **(Time to completion minimized):**

$$T = \frac{MMn}{N} + \frac{MMc}{2}N$$

$$\frac{dT}{dN} = -MMn \times N^{-2} + \frac{MMc}{2} \quad (set\ to\ 0\ \ for\ minimum T)$$

$$-MMn \times N_{opt}^{-2} + \frac{MMc}{2} = 0$$

$$N_{opt} = \sqrt{\frac{MMn}{\frac{MMc}{2}}} = 1.41\sqrt{\frac{MMn}{MMc}}$$

- **Example : A 100 Man month task requires an average of 1 Man Month Communications effort per pair of project personnel. Nopt = ?**

$$N_{opt} = 1.41\sqrt{\frac{MMn}{MMc}} = 1.41\sqrt{\frac{100}{1}} = 14.1\ Persons$$

$$T_{opt} = \frac{100}{14.1} + \frac{1}{2} \times 14.1 = 14.14\ Months$$

$$\sum MM = T_{opt} \times N_{opt} = 200\ Man\ Month$$

- **With the 100 MM task effort est., wouldn't it be a shocker if no consideration of communication was made !**

- ## Chpt 3: The Surgical Team
- **A lot of the program managers believe that they are better off having a very small number of sharp Programmers on their team than having a large number of mediocre programmers.**
- **However, the problem with an organization like this is that with such a small number of people in a group you cannot do large system applications and system programs.**
- **For example, the OS360 took about 5,000 MAN-YEARS to complete so if we have a small team of 200 programmers it would have taken them 25 years to complete the project!**
- **However, it took about 3 years to complete the project and, at times, there were more than 1,000 men working on the project.**
- **Now assume that instead of hiring the 200 programmers we hire 10 very smart and productive programmers, therefore, let's say they can achieve an improvement factor of 7 in terms of productivity for programming and documentation, and we can also achieve an improvement factor of 7 in terms of reduced communication among the programmers.**
- **In that case, for our 5,000 MAN-YEARS job, we need 10 years to complete the project. (5,000/(10X7X7)).**
- **Solution:**

- Large programming jobs must be broken into smaller subtasks and each programmer will work on a subtask.
- But as a whole, the team must act like a surgical team, in other words, there would be a team leader or chief programmer (surgeon), who is in charge of the whole thing and other team members try to help him complete the project (surgery).
- Chief Programmer (and Co-Pilot)
  - Administrator (handles money, people, space, machines, etc.)
    - Secretary
  - Editor (Supports Chief Programmers Documentation preparation)
    - Secretary
  - Program Clerk (Maintains code and all technical records)
  - Toolsmith (Serves the chief programmer's needs for utilities, proc's, macros, etc.)
  - Tester (Devises system component tests from the functional spec's, Assists in day to day debugging, etc.)
- Difference between a surgical team and a team of collaborating colleagues:
  - In the surgical team, the whole project is the brainchild of the surgeon and he/she is in charge of the project. The surgeon (or chief programmer) is in charge of keeping the conceptual integrity of the project.
  - In a collaborative team, on the other hand, everyone is equal and everyone can have input into how the project should change and how the integrity should be preserved, and that sometimes causes chaos.

- **Scaling up: How does one apply the surgical team organization to a project which involves hundreds or thousands of team members?**
  - **The obvious choice is to use a hierarchical organization in which the large system is broken into many subsystems and there are surgical teams assigned to each subsystem.**
  - **Of course, there has to be coordination between the subsystems and that will be as coordination between the subsystem team leaders (a few chief programmers).**

- **Chpt 4: Conceptual Integrity**
- **Conceptual integrity is probably the most important aspect of large system programming.**
- **It is better to have one good idea and carry it through the project than having several uncoordinated good ideas.**
- **It is important that all the aspects of the system follow the same philosophy and the same conceptual integrity throughout the system. Thus, by doing so, the user or the customer only needs to be aware of or know one concept.**
- **The conceptual integrity of the whole system is preserved by having one system architect who designs the system from top-to-bottom.**
- **It should be noted that the system architect should stick with the design of the system and should not get involved with the implementation.**

- This is because implementation and design are two separate phases of the project.
- When we talk about the architect and architecture of a system we are referring to the complete and detailed specifications of all the user interfaces for the software system.
- The implementation of the system, on the other hand, is what goes on behind the scenes, and what is actually carried out in coding and in programming.
  - An example is a programming language such as FORTRAN. The architecture of FORTRAN specifies the characteristics of FORTRAN, its syntax and its semantics. However, we can have many implementations of FORTRAN on many different machines.
- A system architecture is based on user requirements.
- An architect defines the design spec's while a builder defines the implementation spec's.
- How is the architecture enforced during the implementation?
  - Manual is the chief product of the architect and it carries the design spec's.
  - Meetings to resolve problems and discuss proposed changes.
  - Product testing.

Chpt 7: Why Did the Tower of Babel Fail

- Originally, the people of Babel spoke the same language and they were all united.
- Therefore, they were successful in initiating and implementing the Tower of Babel to reach God.
- However, later, God made them speak different languages and the project failed!
- The Tower of Babel failed not because of technological limitations or lack of resources. It failed because of communication problems and disorganization that followed because of lack of communication.
- The same problem can be applied to large software engineering projects. Many large system programs fail because of communication problems among the programmers and among the implementers.
- How does one cope with the communication problem?
- Besides telephone calls, meetings, E-mail and other forms of communication, keeping a workbook for the project is extremely important.
- The workbook essentially contains all the documentation relevant to the project (Objectives, External Spec's, Internal Spec's, etc.).
- It especially contains documentation about the changes made to the project as the project progresses.

- This will be a tool where the team members can go to find out what the newest changes are and what other people are doing.
- It should be noted that, as the projects become larger and larger, and the number of people involved become more and more, the size of the project workbook increases and we have to impose some kind of a structure for the workbook. For example, we may use a tree structure organization, or we may have to use indexing and numbering systems in order to find what we are looking for.
- One of the problems to deal with is how to keep track of changes of the project in the workbook. There are two things that need to be done:
  - 1) All the changes should be clearly highlighted in the documents and this can be done, for example, by a vertical bar in the margin for any line which has been changed.
  - 2) There should be a brief summary of changes attached, which highlights or explains briefly the significance of the changes done to the project.
- Organization:
  - Organization is a means to control communication in a project.
  - We have already talked about tree organization, as well as matrix organization, and also we have talked about the communication involved in each structure.
  - For example, in the tree organization we see a clear division of responsibilities and the communication becomes limited among subordinates and the supervisors.

- **Chpt 11: Plan to Throw One Away**
- **Building a pilot model or a pilot system is an intermediate step between the initial system implementation (or prototyping) and the final large-scale system.**
- **Building a pilot model is necessary in order to overcome the difficulties unforeseen when going from a small-scale project to a large-scale project.**
- **The pilot system should be thrown away and should not be given to the customer as a final model. This is because the pilot model often has bugs in it and would reduce the customer's confidence in the producer.**
- **Software Maintenance:**
  - **The major difference between software maintenance and hardware maintenance is that in hardware maintenance we simply repair or replace components and the architecture and the interface to the user doesn't change.**
  - **However, in software maintenance we make design changes to the software and typically add new functions and because of this, the interface to the user changes.**
  - **Also, in software maintenance, fixing the problems and new functions also introduce new bugs which requires further testing.**
  - **What typically happens is that as we add more functions, we are adding more bugs which in turn requires more fixes and modifications. Therefore, software maintenance is a very difficult job.**

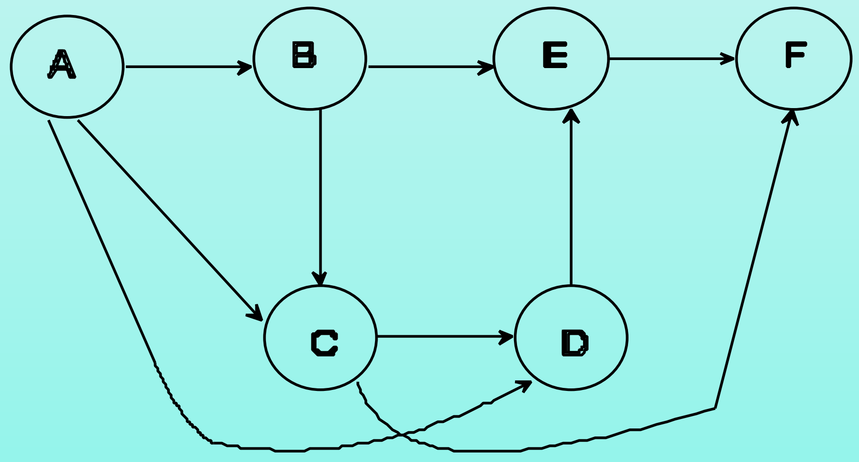
- **Chpt 14: Hatching a Catastrophe**
- **Milestones must be clearly defined, very sharp and easily identifiable.**
- **PERT charts are very important in keeping track of the critical tasks in the whole project.**
- **By identifying the critical path in a PERT chart we know which tasks cannot afford even one day of delay in their completion.**

- **Chpt 15: The Other Face**
- **Read chapter 15 for details on how to document a large program.**

SIMPLIFIED PERT

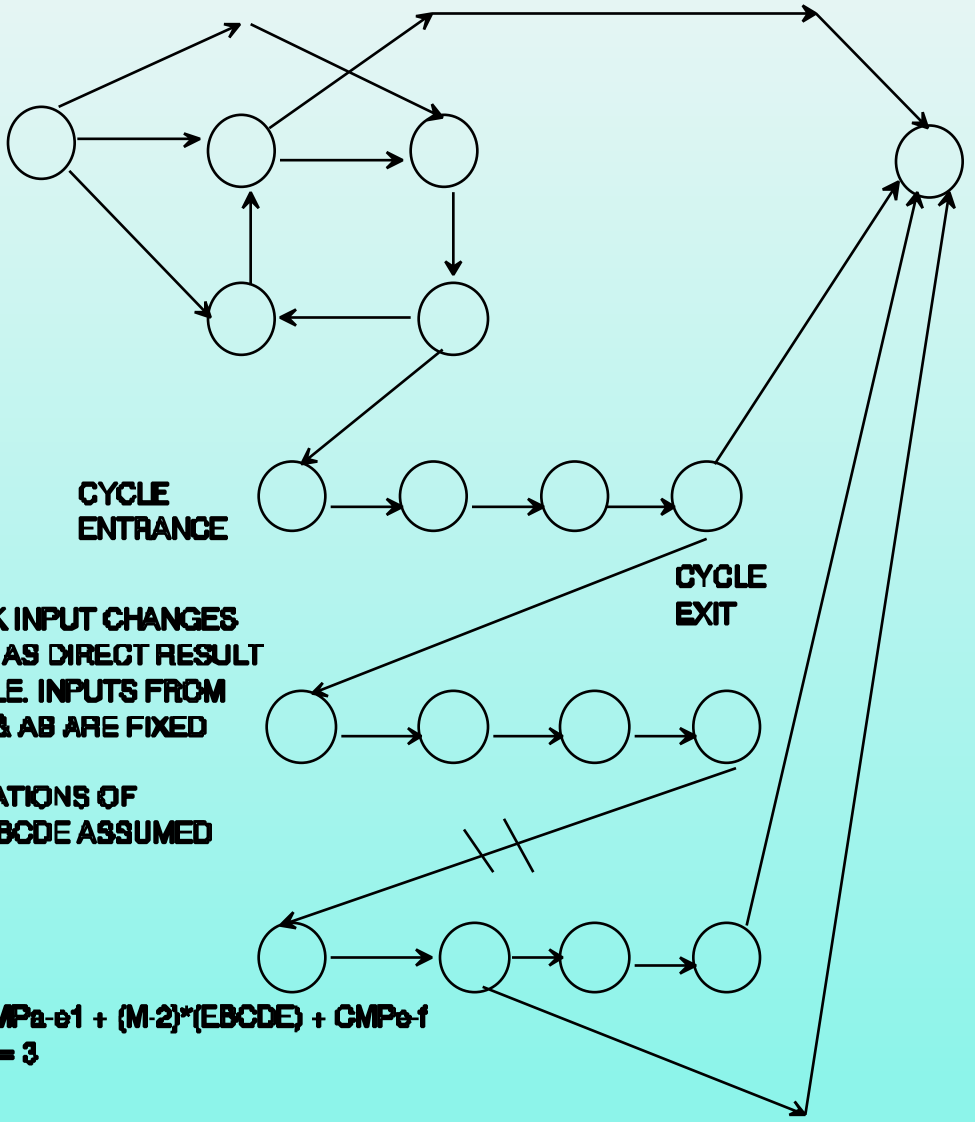
TASK	A-B	A-C	A-D	B-C	B-E	C-D	C-F	D-E	E-F
TIME	2	5	6	4	4	1	7	2	8

- CMP ?
- SLACK TIME ?



Change B-E to E-B ?

HANDLING CYCLIC TASKS



CYCLE
ENTRANCE

CYCLE
EXIT

NO TASK INPUT CHANGES
EXCEPT AS DIRECT RESULT
OF CYCLE. INPUTS FROM
AC, AD, & AB ARE FIXED

M ITERATIONS OF
CYCLE BCDE ASSUMED

CMPa-f = CMPa-e1 + (M-2)*(EBCDE) + CMPe-f
ASSUME M = 3

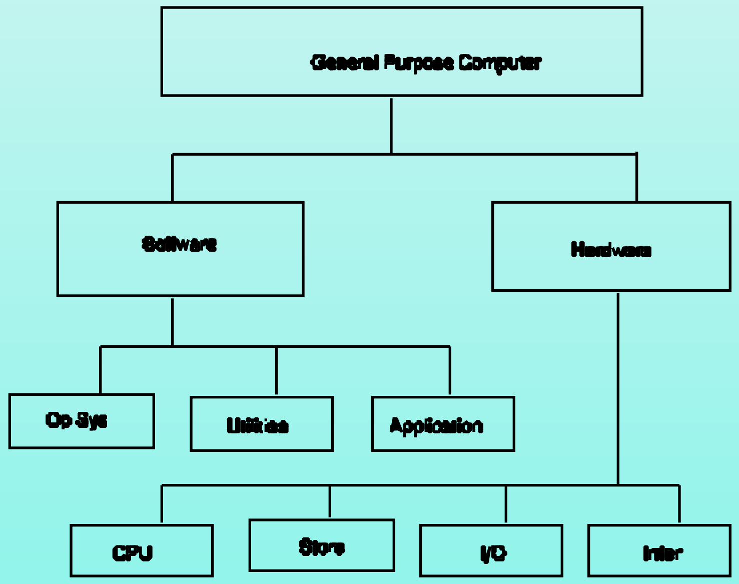
DESIGN METHODOLOGIES

- Design Notations :
  - Software :
    - » Data Flow Diagrams
    - » Structure charts
    - » Procedure Templates
    - » Pseudocodes
    - » Structured Flowcharts
    - » Structured English
    - » Decision Tables
  - Hardware
    - » Block Diagrams
    - » Schematic and wiring diagrams
    - » Layout diagrams
    - » Detailed drawings (FAB, detail, PCB masks)

- **Design Strategies (software and hardware)**
  - No generally applicable recipes.
  - Conceptual gap : System requirements ----> Primitive bldg blocks
  - Many design alternatives
  - (S/W): Space - Time trade offs. (H/W): thermal , Structural & Maintenance access analysis.
  - Iteration
  - Divide and Conquer : Partition the problem into smaller manageable pieces.
- **Design Techniques (S/W) :**
  - Stepwise refinement
  - Levels of Abstraction
  - Structured Design
  - Integrated Top Down Development
  - Bottom Up development
  - Jackson Structured Programming

DESIGN TECHNIQUES

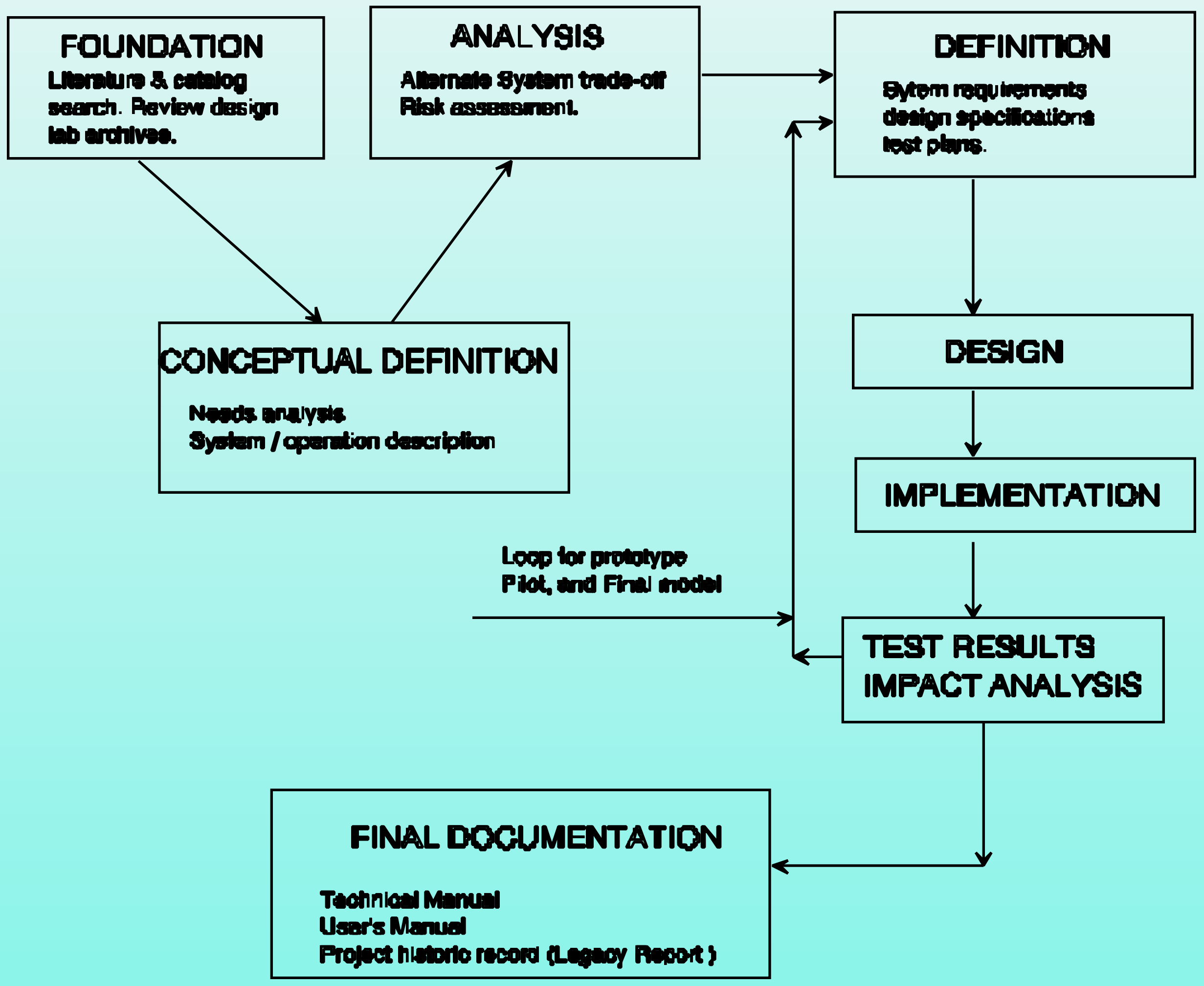
- **TOP DOWN APPROACH**
  - Strong system requirements
  - The successive decomposition of a large problem into smaller sub problems until suitable primitive building blocks are reached.



DESIGN TECHNIQUES

- BOTTOM UP APPROACH
  - Restricted resources and weak system requirements.
  - The basic primitives are used to build useful and more powerful problem oriented building blocks.

THE DESIGN PROJECT PROCESS


FOUNDATION
Literature & catalog search. Review design lab archives.
ANALYSIS
Alternate System trade-off Risk assessment.
DEFINITION
Sytem requirements design specifications test plans.
CONCEPTUAL DEFINITION
Needs analysis
System / operation description
DESIGN
IMPLEMENTATION
Loop for prototype Pilot, and Final model
TEST RESULTS IMPACT ANALYSIS
FINAL DOCUMENTATION
Technical Manual
User's Manual
Project historic record (Legacy Report)