09/08/17

Software Engineering

* **Software Engineering** → systematic and cost-effective techniques for software development.
  - Past experience is used. Theoritical basis has been provided where possible, otherwise, experience is adopted.
    On the contrary, scientific solutions are constructed through rigorous application of parallel principles.
  - Several alternate solutions are tired. Iterations and backtracking are used.
    Science, on the other hand, uses only unique solutions.
  - Engineering disciplines use only well-documented and well-understood principles.
    Arts, on the other hand, uses subjective judgement.
  - Exploratory software development → FAILS for large software. Programmers uses his intuition and experience.

* Principles deployed by Software Engineering to overcome human cognitive limitations.
  (i) Abstraction (eg Modelling); and
  (ii) Decomposition (Divide and conquer)

Software LIFE-CYCLE Model → a well-defined and ordered set of activities, typical of software engineering approach to software development.

1. Feasibility Study ↓
2. Requirements analysis and specification ↓
3. Design ↓
4. Coding and unit testing ↓
5. Integration & system testing ↓
6. Maintenance

② Requirements analysis and specification:

- Gather requirements from customer.
- Analyze (to remove inconsistencies and incompleteness)
- Prepare SRS (Software Requirements Specification) document. SRS serves as a contract between the development team and the customer. It is the basis of all development activity. The customer must understand it.

~~③ Design → Software architecture is derived from the SRS document~~

① Feasibility study → To determine whether it would be financially and technically feasible to develop the software.

- Develop an overall understanding.
- Formulate various possible strategies.

③ Design → Software architecture is derived from the SRS document.

Goal: To transform the requirements specified in the SRS document into a structure that is suitable for implementation in some programming language.

Two approaches:

(i) Procedure design approval.
(ii) Object-oriented design approach.

④ Coding and unit testing

Purpose to translate a software design into source code and to ensure that individually, each function is working correctly. End product of this phase → a set of program modules that have been individually unit tested.

testing, debugging to fix problems, and ~~manager~~ management of test cases.

(5) Integration and system testing:

Integration of various modules is normally carried out incrementally over a number of steps. Integration testing is carried out to verify that the interfaces among different units are working satisfactorily.
Finally, after all modules have been successfully integrated and tested, the full working system is obtained.

SYSTEM TESTING: is carried out on the fully working system. (Goal: to ensure that the system conforms to the SRS).

(6) Maintenance:

Maintenance: 60% effort
Development: 40% effort

- ~~Consecutive~~ Corrective maintenance → to correct errors that were not discovered during the product development phase.
- Perspective maintenance → to improve the performance of the system, or to enhance the functionalities of the system.
- Adaptive maintenance → for porting the software to work in a new environment.

SHORTCOMINGS of the classical waterfall Model:

(1) No feedback path → each phase, once closed, can't be reopened. So every phase must be carried out flawlessly. This is idealistic (Errors do occur at every stage)

(ii) Difficult to accommodate change requests:
This model assumes that all customer requirements can be completely and correctly defined at the beginning of the project. This is hard to achieve. Customer's requirements usually keeps on changing with time.

(iii) Inefficient error corrections:
In this model, integration of code and testing are done very late. At that stage, problems are harder to resolve.

(iv) Overlapping of phases:
For efficient utilization of manpower, phases should overlap. For example, design of ~~the~~ test cases can overlap with design.

22/08/19 Prototyping Model:

Approach: Build a working prototype BEFORE development of actual software.

Applicability:
(i) development of GUI.
(ii) The GUI part of a software system is almost always developed using the prototyping model.
(ii) When the development team has very little knowledge of the technical issues involved.
(iii) when development of highly optimized and efficient software is required.

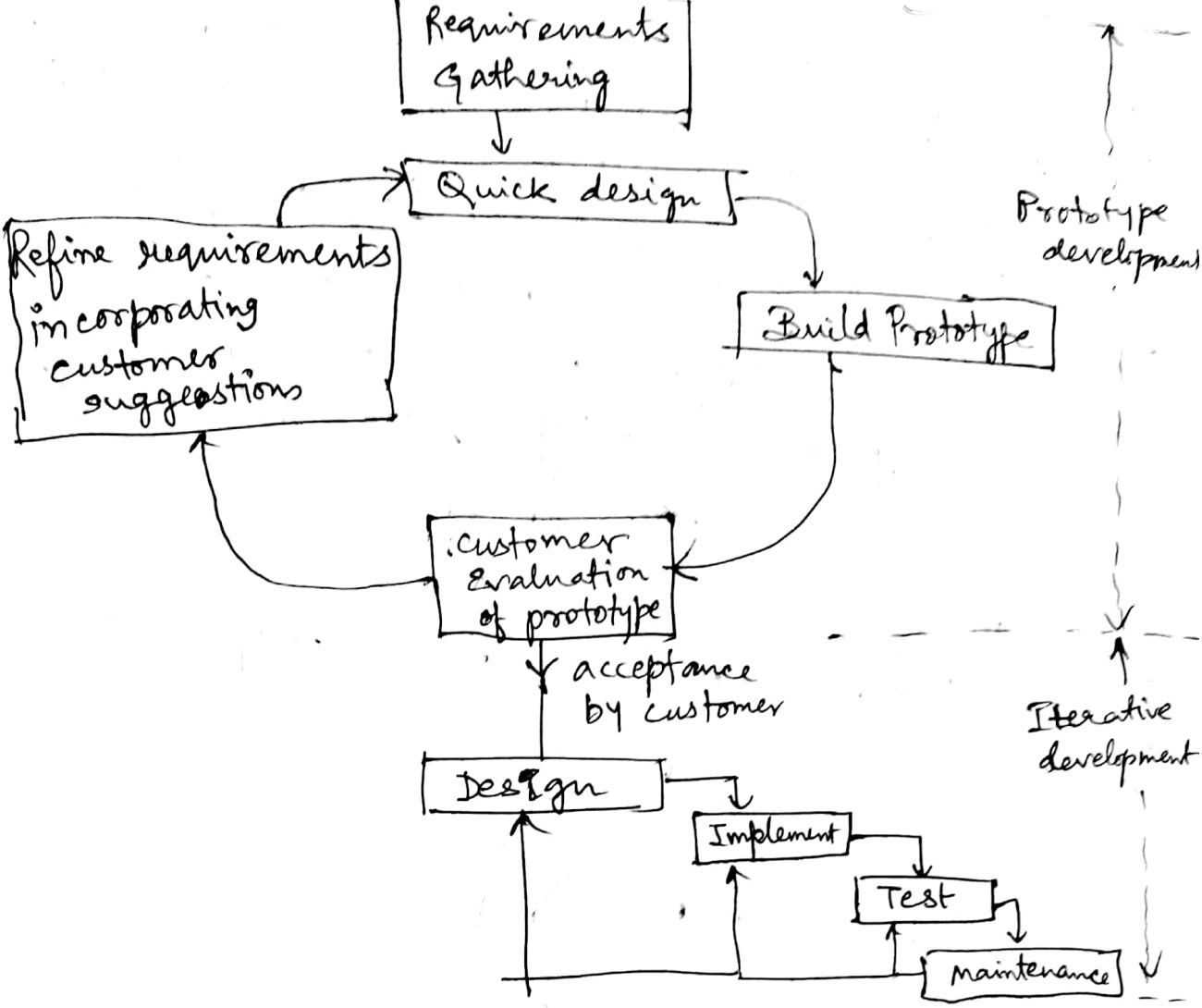
This ~~model~~ involves the major activities
(A) Prototyping construction (this is later thrown away)
(B) Iterative waterfall based Software development.

Refine req
incorpora
custom
sugg

Weakne
This
which
the

Rapid
• Co
Th
de
• It
re
(e
•
(

•



Weakness:

This model is effective only for those projects for which the risks can be identified upfront before the project starts.

Rapid Application Development (RAD) model:

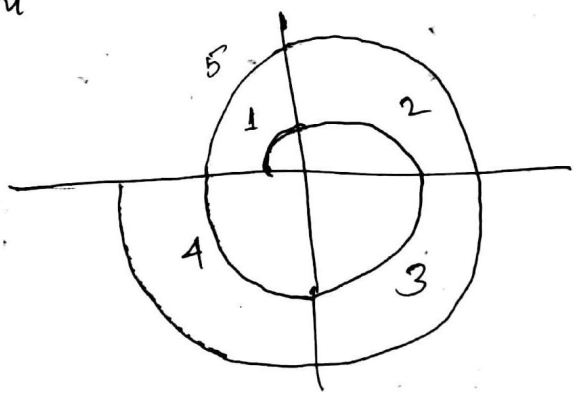
- Code reuse is advocated. Thus to use RAD, a company should have had developed similar software earlier.
- It is not suitable for cases where optimum reliability (e.g. Operating System) or performance (e.g. Height Simulator) is required.
- It is suitable for customized software (developed for one or two customers by adopting an existing software).
- It is suitable for projects with very aggressive time schedules.

How it works:

RAD takes place in a series of short cycles or iterations. Each iteration is planned to enhance the implemented functionality of the application by only a small amount. A prototype is built for the functionality. The customer evaluates it and gives his feedback. The prototype is then developed. (In contrast, the prototype life-cycle model throws away the prototype).

SPIRAL model:

Applicability: when unknown risks crop up during development



- One cycle (traversal through 4 quadrants) is called a phase. During a phase, some features of the software is added.
- In each phase, risks are evaluated and a prototype is developed.
- With each iteration around the spiral, progressively more complete versions of the software get built.

1. Determine objectives, alternatives, and constraints
   - alternatives → build, reuse, buy, subcontract
   - constraints → cost, schedule, interface, environmental limitations.
   - objectives → performance, functionality, ability to accommodate change, hardware/software interface, critical success factors

Risks associated with lack of experience new technology. tight schedules, poor processes and so on, are documented.

2. Evaluate alternatives and identify and resolve risks. Solutions are evaluated by developing an appropriate prototype.

3. Develop next-level product: → at the end of the quadrant, the identified features have been implemented and the next version of the software is available.
- Creation of design
- Renew of design.
- Development of code.
- Inspection of code.
- Testing and packaging of product.

4. Plans next phase
- Development of project plan.
- Development of configuration management Plan.
- Development of test plan
- Development of installation plan.

[In this quadrant, the team reviews the developed version of the software with the customer and plans the next iteration of the spiral].

Spiral Model	Prototyping Model
• can handle unforseen risks that show up much after the project has started.	• Risks must be identified Before development starts.

SOFTWARE PROJECT PLANNING:

It encompasses 5 major activities:

1. Estimation.
2. Scheduling
3. Risk analysis.
4. Quality management planning.
5. Change management planning.

Software Cost and Effort Estimation:

Methods.

1. Decomposition techniques.
2. Empirical estimation models.

→ 1. By decomposing a project into major functions and related software engineering activities, cost and effort estimation can be performed in a stepwise fashion.

SOFTWARE SIZING:

1. Direct approve — size measured in lines of code (LOC).
2. Indirect approve — size represented as functio points. (FP).

F.P. based Estimation:

The function point metric (FP) can be used effecti as a means of measuring the functionality delivered by a system.

Measures of Information Domain:

1. Number of external inputs (EIs)
2. Number of external outputs (EOs).
3. No. of external inquires (EQs)
4. No. of internal logical files (ILFs)

5. No of external interface files (EIFs).

Information domain values.	Count	weighting factor simple	Avg	Complex
External Inputs (EIs)		3	4	6 =
External Outputs (EOs)		4	5	7 =
External Inquiries (EQs)		3	4	6 =
Internal Logical Files (ILFs)		7	10	15 =
External Interface files (EIFs)		5	7	10 =
Count total (Raw fn point count)				

FP = (count-total) × [0.65 + 0.01 × Σ(Fi)]

The Fi (i=1 to 14) are value adjustment factors

Value adjustment Factors (VAF):

The Fi (i = 1 to 14) are based on responses to the following questions:

1. Does the system require reliable backup and recovery?
2. Are specialized data communications required to transfer information to or from the application?
3. Are there distributed processing functions?
4. Is performance critical?
5. Will the system run in an existing heavily utilized operational environment?
6. Does the system require on-time data entry?
7. Does the on-time data entry require the input transaction to be built over multiple screens or operations?
8. Are the ILFs updated on-line?
9. Are the outputs, files, inputs or inquiries complex?
10. Is the internal processing complex?
11. Is the code designed to be reusable?
12. Are conversion and installation included in the design.

13. Is the system designed for multiple installation in diff. organizations.

14. Is the application designed to facilitate change and for ease of use by the users.

SCALE: 0 – 5

↙ Not imp. ↘ Absolutely essential.

06/09/19

Function Points

Example

Information Domain Value :	Opt	Likely	Pess	Est. Count (Rounded off)
No. of external inputs	20	24	30	24
No. of external outputs	12	15	22	16
No. of external inquiries	16	22	28	22
No. of internal logical files	4	4	5	4
No. of external interface files.	2	2	3	2

Count total

Est. count = (Opt + 4 × likely + Pess)/6

FP count = weight × est. count

Value adjustment factor

	Value
(1)	4
(2)	2
(3)	0
(4)	4
(5)	3
(6)	4
(7)	5
(8)	3
(9)	5
(10)	5
(11)	4
(12)	3
(13)	5
(14)	5

$\Sigma F_i = 52$

FP estimated = count-total × [0.65 + 0.01 × ΣF_i]

= 320 × [0.65 + 0.01 × 52]

= 375.

Constructive COst MOdels (cocomo).

- COCOMO was originally described in the book:
  Software Engineering Economics
  — by W.B. BOEHM. Prentice Hall, 1981.
- Boehm developed his cost estimation system by carefully studying the data related to 63 completed software projects. The size of the projects he studied ranged from small (5000 SLOC) to very large (over 1,000,000 SLOC).
  [SLOC = source lines of code].
- COCOMO consists of 3 levels of model precision:
  * basic
  * intermediate
  * detailed.

(Rounded off) ss	Est. Count	weight	FP count
0	24	(24.33) 4	97
2	16	(15.67) 5	78
8	22	(22) 4	88
5	4	10	42
3.	2	7	15
count total			320.

- Labor months (LM) and development time (DT) are estimated. (LM) ↑ cost estimate; (DT) ↑ schedule estimate.

cost (in $) = labor months × avg. labor cost/month

Basic COCOMO Model

The basic COCOMO distinguishes 3 software development modes:

- Organic
- semidetached
- embedded.

Organic Mode:

- The development team consists of a small number of experienced developers who are familiar with both the application area and the

— 5
— 5
$\Sigma F_i = 52$
otal x
5 + 0.01 x ΣF_i]
[0.65 + 0.01 x 52)

development environment.

- The product size ranges from small to mediu

$$LM = 2.4 \times (KSLOC)^{1.05}$$
$$DT = 2.5 \times (LM)^{0.38}$$

[KSLOC = estimated size of the project measured in thousands of source lines of equivale executable new code.].

Software Requirements Specification (SRS). IEEE 830-1998

Title Page

<Name of project>.
<author>
<date>.

Version	Release date.	Responsible party	Major changes

Table of contents

Section	Description.	Page
1	Introduction	
2	General Description	
3	System Requirements	
4	Supporting Information	

1. Introduction.

1.1 Purpose: Identify the purpose of the SRS and its intended audience.

1.2 Scope:
1. Identify the software product to be produced.
2. Explain what the software product will, and if necessary will not do.

3. Describe the application of the software.
as part of this.
a. Describe the relevant benefits, objectives, and goals as precisely as possible.
b. Be consistent with similar statements in higher-level specifications, if they exist.

1.3 Definitions, acronyms, and abbreviations may be provided by reference to an appendix.

1.4. References:
1. Provide a complete list of all documents referenced everywhere in the SRS.
2. Identify each document by title, report no, date, and publishing organisation.
3. Specify the sources from where the references can be obtained.

1.5 overview:
Describe the rest of the SRS and how it is organized.

2. General description:
2.1 Product perspective: This subsection relates the product to other products or projects.
If the SRS defines a product that is a component of a larger system or project.
a. Describe the function of each component of the larger system or project, and identify interfaces.
b. Identify the principal external interfaces of this software product (not detailed)

c. Describe the computer hardware and peripheral equipment to be used (overview only).

2.2. Product Functions.

Provide a summary of the functions that the software will perform. (list of functions should be understandable to the customer reading the document for the first time).

2.3. User characteristics.

Describe the general characteristics of the end-users of the product that will affect the specific requirements. (education, experience, technical expertise).

2.4 General Constraints.

e.g. regulatory policies, hardware limitations (e.g. signal timing requirements), interfaces to other applications, parallel operation, audit functions, criticality of the application.

2.5. Assumptions or dependencies

e.g. a specific OS.

SRS (contd.)

3. Specific Requirements

(This is the largest and most important part)

3.1 Functional Requirements

For each function, specify requirements on input, processing, and output.

(i) Purpose of the function.

(ii) Inputs: sources, valid range of values, any timing concerns, operator requirement, special interface.

(iii) Operation to be performed — validity check, response to abnormal conditions, etc.

(iv) Outputs: destination, valid range of values, timing concerns, handling of illegal values, error messages, interfaces required.

3.2 External Interface Requirements

3.2.1 User Interfaces

a. Required screen formats.
b. Page layout and content of any reports or menus
c. Relative timing of inputs and outputs.
d. Availability of some form of programmable function keys.

All the aspects of optimizing the interface with the person who must use the system. (do's and don'ts on how the system will appear to the user).

~~3.3 Performance Requirements~~

3.2.2 Hardware interfaces

Specify the logical characteristics of each interface between the software product and the hardware components of the system. Include such matters as what devices are to be supported, how they are to be supported, and protocols. A block diagram showing the relationship among the hardware blocks and the software function.

hosted in each block is essential here.

3.2.3 Software Interfaces

Specify the use of other required software products (for example, a data management system, an operating system, or a mathematical package), and interfaces with other application systems. For each required software product, the following should be provided:

1. Name.
2. Mnemonic
3. Specification number.
4. Version number.
5. Source

For each interface:

1. Discuss the purpose of the interfacing software as related to this software product.
2. Define the interface in terms of message content and format. (Reference to the document defining the interface is required).

3.2.4 Communication Interface

Specify the various interfaces to communication, such as local network protocols.

3.3 Performance Requirements

1. Static numerical requirements may include
   a. The number of terminals to be supported.
   b. The number of simultaneous users to be supported.
   c. The number of files and records to be handled.
   d. The sizes of files and tables.

2. Dynamic numerical requirements may include, for example, the number of transactions and task and the amount of data to be processed within certain time periods for both normal and peak workload conditions.

3.4 Design Constraints.

- Imposed by other standards, hardware limitations, etc.

3.5 Quality Characteristics :

Correctness, efficiency, flexibility, integrity, interoperability, maintainability, etc.

3.6 Other Requirements

3.6.1 Databases

Requirements for any database that is to be developed as part of the product. These might include.

1. Types of information.
2. Frequency of use.
3. Accessing capabilities.
4. Data element and file description.
5. Relationship of data elements, records and files.
6. Static and dynamic organization.
7. Retention requirements for data.

3.6.2 Operation

Normal and special operations required for each user, e.g.

1. Various modes of operation in the user organisation eg. user-initiated operation.

2. Periods of interactive operations and periods of unattended operation.
3. Data processing support functions.
4. Backup and recovery operations.

3.6.3 Site Adaptation Requirements

1. Define the requirement for any data or initialization sequences that are specific to a given site, mission or operation mode (e.g. safety limits).
2. Specify features that should be modified to adapt the software to an installation.

4. Supporting Information.

Table of contents, Appendices, Index, etc

Section-3

FORMAT - 1

(all functional requirements specified first).

3. Specific Requirements.

3.1 Functional Requirements
- 3.1.1 Functional Requirement 1
  - 3.1.1.1 Introduction
  - 3.1.1.2 Inputs
  - 3.1.1.3 Outputs
- 3.1.2 Functional Requirement 2
- ---
- 3.1.n Functional Requirement n

3.2 External Interface Requirements.
- 3.2.1 User Interfaces.
- 3.2.2 Hardware Interfaces.
- 3.2.3 Software Interfaces.
- 3.2.4 Communication Interfaces.

3.4.2 Hardware Limitations

3.5 Quality characteristics
3.5.1 Correctness
3.5.2 Efficiency
---- etc. ----

3.6 Other Requirements
3.6.1 Database
3.6.2 Operations.
3.6.3 Site Adaptation
- - -

FORMAT-2
(Each Interface Requirement applied to each functional requirement first)

3. Specific Requirements
3.1 Functional Requirements
3.1.1 Functional Requirement 1
3.1.1.1 Specification
3.1.1.1.1 Introduction
3.1.1.1.2 Inputs.
3.1.1.1.3 ~~Outputs~~ Processing
3.1.1.1.4 Outputs.
3.1.1.2 External Interfaces
3.1.1.2.1 User Interfaces.
3.1.1.2.2 Hardware Interfaces
3.1.1.2.3 Software Interfaces
3.1.1.2.4 Communication Interfaces.
3.1.2 Functional Requirement 2
- - -
3.1.n Functional Requirement n
3.2 Performance Requirements.

3.3 Design constraints.

3.4. Quality characteristics

3.5. Other requirements.

FORMAT -3

(A functional requirement, the rest of the Requirem and all Interface Requirements)

3. Specific Requirements.

3.1 Functional Requirements

3.1.1 Functional Requirement 1

3.1.1.1 Introduction.

3.1.1.2 Inputs.

3.1.1.3 Processing.

3.1.1.4 Outputs

3.1.1.5 Performance Requirements.

3.1.1.6 Design constraints.

3.1.1.6.1 Standards Compliance

3.1.1.6.2 Hardware Limitations.

3.1.1.7 Attributes.

3.1.1.7.1 security.

3.1.1.7.2 Maintainability.

3.1.1.8 Other Requirements

3.1.1.8.1 Database.

3.1.1.8.2 Operations.

3.1.1.8.3 Site Adaptation.

3.1.2 Functional Requirement 2

:

3.1.n Functional Requirement n.

3.2 External Interface Requirements.

3.2.1 UI

~~3.2.2 Hardware Interfaces.~~

6. Sour.
20. Anusan
11. Radib Kar.
10. Shesrata Saha
Vishal Kanti Ghosh.
Guha
Haldar. } 17/09/1..

3.2.1.1. Performance Requirement
3.2.1.2 Design Constraints
3.2.1.2.1 Standards Compliance.
3.2.1.2.1 Hardware Limitations.

3.2.1.3. Quality characteristics

3.2.1.4 Other requirements.
3.2.1.4.1 Database.
3.2.1.4.2 Operations.
3.2.1.4.3 Site Adaptations.

3.2.2 Hardware Interfaces.
3.2.3 Software Interfaces
3.2.4 Communication Interfaces.

PROJECT SCHEDULING :

Select a process Model
↓
Identify software engg. tasks to be performed
↓
Estimate amount of work & no. of people.
↓
Consider the risks.
↓
Create a network of software engg tasks to get the job done on time.
↓
Assign responsibility for each task.

WHY:
To build a complex system, many software engg tasks occur in parallel; and the result of work performed during one task may have a profound effect on work conducted in another task.

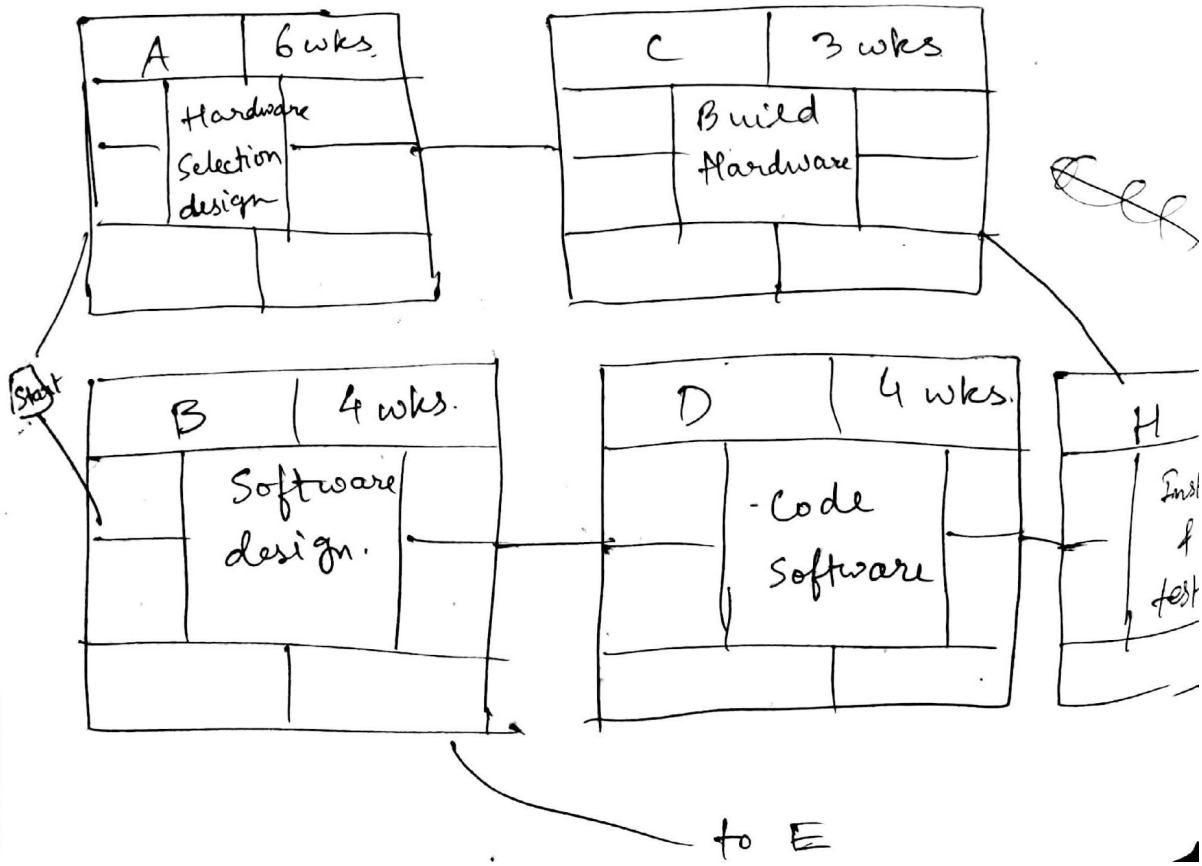
Scanned by CamScanner

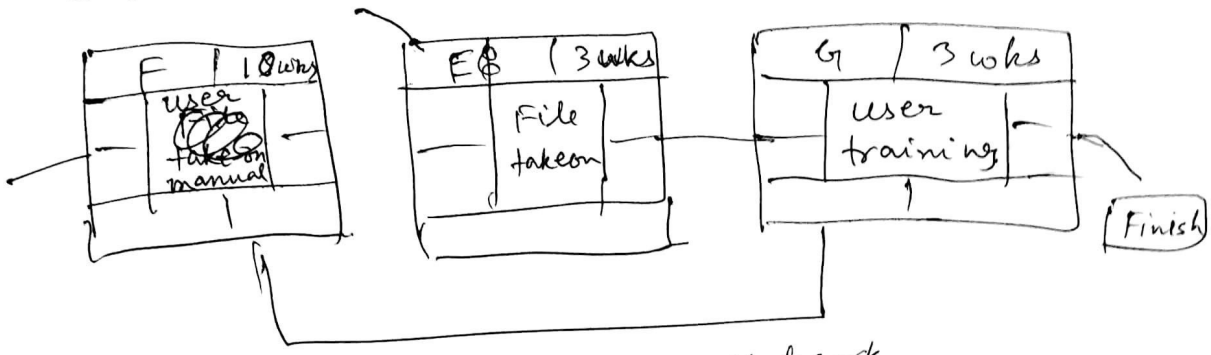
These interdependencies are very difficult to understand without a schedule. It is also virtually impossible to assess progress on moderate or large software project without a detailed schedule.

Project Specification

Activity	Duration (week
A. Hardware selection	6
B. Software design.	4
C. Build Hardware.	3
D. Code and test software.	4
E. File take-on.	3
F. Write user manuals.	10
G. User training.	3
H. Install and test system.	2

Network Model





Precedence Network.

Precedents.

A

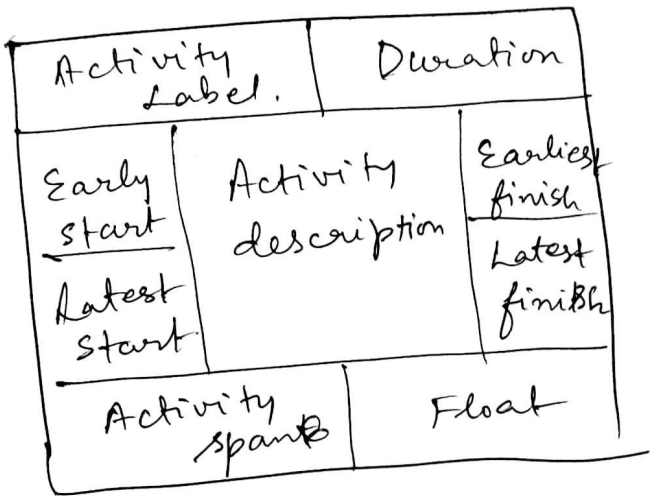
B

B

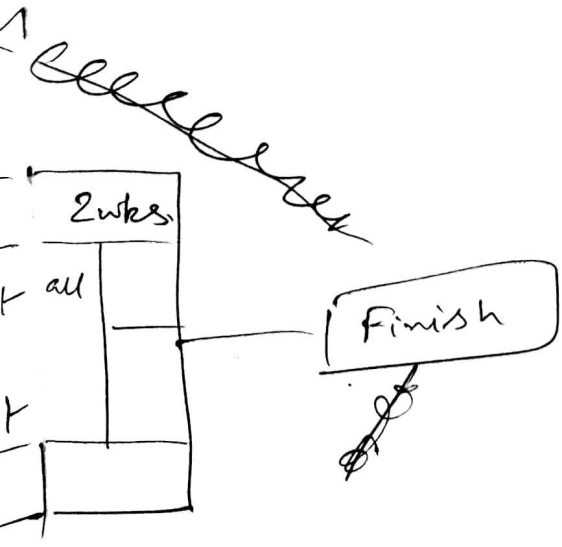
.

E, F

G D.



Labelling Convention.



$$\forall(x)\ (set(x) \Rightarrow \exists(y)\exists(u)\exists(v)(set(y) \wedge card(x,u) \wedge card(y,v) \wedge greater(v,u)))$$

$$\forall(x)\ (set(x) \Rightarrow \exists(y)\ \exists(u)\ \exists(v)(set(y) \wedge card(x,u) \wedge card(y,v) \wedge greater(v,u)))$$

Machine learning 27/9/19.

~~Information gain, Chi-Square~~

If a dataset is given with N no. of features, first calculate the following measures in order to identify important features :-

a) Information gain

b) chi square

Again try to find out which features can be gr... to make a good model for classification (you can ... any correlation measures for identifying the model).

Qns are :-

1) Identify the features that are less important with support and confidence < 60%.

2) Identify the missing values from the dataset. and normalise them accordingly

3) Implement the following algorithm to classify the instances with respect to most important features ($M < N$), and the best model

Algorithm 1 : Decision Tree

" 2 : Naive Bayes

~~3 : If~~

If possible use a k-means clustering algorithm on the dataset without considering class level and verify the results with the other two