Software Engineering

Unit I

Software Process and Planning

* What is software?

→ A product that software professionals build and support over the long term.

→ Software encompasses:

(i) instructions - in the form of computer programs, that when executed provide desired features, function and performance.

(ii) data structures - which enable programs to adequately store and manipulate data.

(iii) documentation - that describes the operation and use of the programs.

characteristics → is engineered
→ doesn't wear out
→ is complex

* Dual Role of Software

(1) As a product
→ delivers computing potential
→ produces, manages, acquires, modifies, displays or transmits information

(2) As a vehicle for delivering a product
→ supports / provides system functionality
→ controls other programs - OS
→ helps communication - networking software

→ helps build other software - software tools

* Software Applications

→ system software - such as compilers, editors, file management utilities

→ application software - standalone software/programs for specific needs

→ engineering / scientific software - characterized by number crunching algorithms such as automotive stress analysis, orbital dynamics etc.

→ embedded software - resides within a product or system, eg. keypad of microwave

→ product-line software - focus on a limited marketplace to address mass consumer market (word processing, graphics, dbms)

→ Web applications - network centric software, web 2.0, remote dbs

→ AI software - uses non-numerical algorithms to solve complex problems (robotics, expert systems, pattern recognition)

* New Categories of Software

(i) Ubiquitous computing - wireless networks, called open world computing

(ii) Netsourcing - the Web as a computing engine, architects simple & sophisticated applications to target end-users worldwide.

(iii) Open Source - with freee source code

(iv) Grid computing, data mining, nanotechnology software

* The Need to change Legacy Software

→ software must be adapted to meet the needs of new computing environments or technology.

→ must be enanced to implement new business requirements

→ must be extended to make it interoperable w/ other system

→ must be re-architected to make it viable within a network environment

* Software Products

A. Generic Products

→ stand-alone systems that are marketed and sold to any customer who wishes to buy them

→ eg. PC software such as editing, graphics programs, CAD etc

B. Customized Products

→ Software that is commissioned by a specific customer to meet their own needs

→ eg. embedded control systems, air traffic control software etc.

* Importance of Software

→ The economies of all developed nations are dependent on software.

→ More and more systems are software controlled in transportation, medical, telecommunications, military etc.

→ Expenditure on software represents a significant fraction of GNP in all developed countries.

* Software Costs

→ Software costs often dominate computer system costs. The costs of software on a PC are often greater than the hardware cost.

→ Software costs more to maintain than it does to develop. For systems with long life, maintenance costs may be several times the development costs.

→ Software engineering is also concerned with cost-effective software development.

* Characteristics of Software (contd. from pg 1)

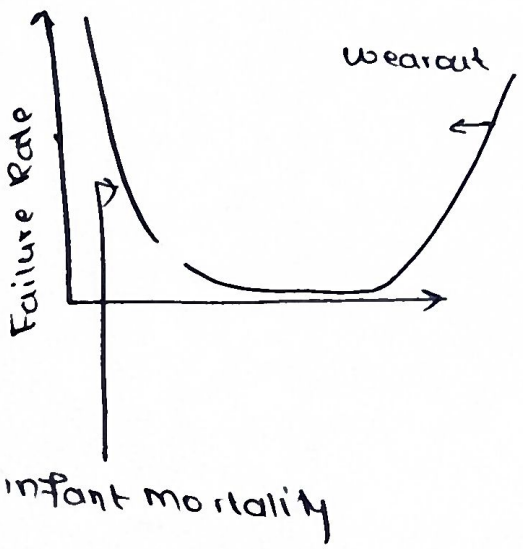
(1) Software is developed or engineered. It is not manufactured in the classical sense, which has quality problems.

Developed	vs.	Engineered
→ High quality is achieved through good design		→ Manufacturing phase for hardware can introduce quality problems
→ Dependent on people		→ There are relationships between different teams of people and the work accomplished is completely different

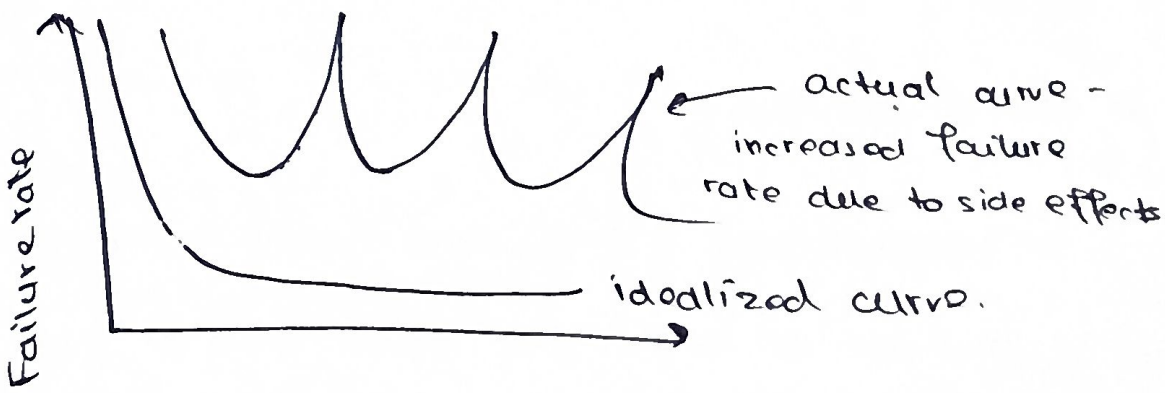
→ Both require the construction of a product, but the approaches used are different.

(2) Software doesn't wear out.

Hardware



Software



(3) Software continues to be custom-built

→ In the hardware world, reuse is a natural part of the engineering process. In the software world, it has only begun to be achieved on a broad scale.

→ A software component should be designed and implemented so that it can be reused in many different programs.

→ Modern reusable components encapsulate data and processing into software parts to be reused by different programs.

eg. GUIs, pull-down menus, interaction mechanisms

* Software Myths

① Management myths :

→ Assuming that people already have everything they need to know, since there already exists books full of standards and procedures for building software.

→ Assuming that if one is behind schedule, one can merely add more programmers to catch up. @alled the Mongolian horde concept.

→ Assuming that outsourcing a software project to a third party is as simple as letting that firm build it.

② Customer Myths

→ Assuming that a general statement of objectives is sufficient to begin writing programs, and that details can be filled in lala

→ Software requirements continually change, but assuming that change can be easily accommodated because software is flexible.

③ Practitioner's Myths

→ Assuming that once the program is written & it works, the job is done.

→ Assuming that until the program runs, there is no way of assessing its quality.

→ Assuming that the only deliverable product for a successful project is the working program.

→ Assuming that software engineering will make one create voluminous and unnecessary documentation, and invariably slow one down.

read slide 24 on ppt 1 for myth examples

* Definition of Software Engineering

A. The seminal definition

Software engineering is the establishment and use of sound engineering principles in order to obtain economical software that is reliable and works efficiently on real machines

B. The IEEE definition.

Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation

and maintenance of software, i.e the application of engineering to software.

* Principles of Software Engineering

1. The Reason It All Exists
2. KISS - Keep It Simple, Stupid!
3. Maintain the vision
4. What you produce, others will consume
5. Be Open to the Future
6. Plan Ahead for Reuse
7. Think

First Principle - The Reason It All Exists

→ A software system exists for one reason - to provide value to its users.

→ Before specifying a system requirement, before noting a piece of system functionality, before determining hardware platforms or development processes - think whether it adds value to the system.

→ If it doesn't, don't do it.

Second Principle : KISS - Keep It Simple, Stupid!

→ All design should be as simple as possible, but no simpler.

→ This facilitates having a more easily understood and easily maintained system.

→ It often takes a lot of thought and work over multiple iterations to simplify.

→ The payoff is software that is more maintainable and less error-prone.

Third Principle - Maintain the Vision

→ A clear vision is essential to the success of a software project.

→ Having an empowered architect who can hold the vision and enforce compliance helps ensure a very successful software project.

Fourth Principle - What You Produce, Others Will Consume.

→ Someone else will use, maintain, document or depend on being able to understand your system.

→ Always specify, design and implement knowing someone else will have to understand what you are doing

Fifth Principle - Be Open to the Future

→ A system with a long lifetime has more value.

→ Software systems must endure far longer. To do this successfully these systems must be ready to adapt to changes.

Sixth Principle - Plan Ahead for Reuse

→ Reuse saves time and effort.

→ The reuse of code and designs has been proclaimed to be a major benefit of using object-oriented technologies.

→ Planning ahead for reuse reduces the cost and increases the

value of both the reusable components and the systems into which they are incorporated.

Seventh Principle - Think!

→ Place clear, complete thought before action to produce better results.

* Software Development Life Cycle Mode (SDLC model)

SDLC - used to describe a process for planning, creating, testing and deploying an information system. The phases of SDLC are:

(i) Preliminary Investigation - includes, goal, objectives, scope of problem, alternative solutions

(ii) Requirement Analysis : requirement definition & specification

(iii) System Design : Layouts, models, pseudo-code etc.

(iv) Implementation : writing code

(v) Integration & Testing : units / components are designed and tested, units are combined into models and testing

(vi) Installation & Maintenance: beta versions, production, upgradation and extensions

(vii) Evaluations: surveys, feedback, statistics

* Capability Maturity Model (CMM)

→ A benchmark for measuring the maturity of an organization's software process

→ A development model created after the study of data collected from organizations that were contracted with the U.S Department of Defense.

→ CMM defines 5 levels of process maturity based on certain Key Process Areas (KPA)

→ The model aims to improve existing software-development processes.

CMM Levels

Level 1 - Initial (~70%)

→ Uncontrolled, undocumented and reactive manner by users or events

→ Provides a chaotic or unstable environment for the processes.

Level 2 - Repeatable (~15%)

→ Some processes are repeatable, with consistent results.

Level 3 - Defined (<10%)

→ These are sets of defined and documented standard processes established and subject to some degree of improvement over time.

Level 4 - Managed - Using process metrics, management can effectively control development process. Strategic analysis performed through data collected on the quality of processes.

Level 5 - Optimizing - Focus is on continually improving process performance through both incremental and innovative technological changes / improvements., along with qualitative feedback.

see ppt @ slide 6 for step diagram w/ the same explanation

* Software Process Models

Models

Prescriptive Process Models

A. Linear Models
- → Waterfall Model
- → RAD Model
- → Incremental Model

B. Iterative Model

C. Evolutionary Model
- → Prototyping Model
- → Spiral Model

D. Concurrent Model

Specialized Process Models

→ Component-based development

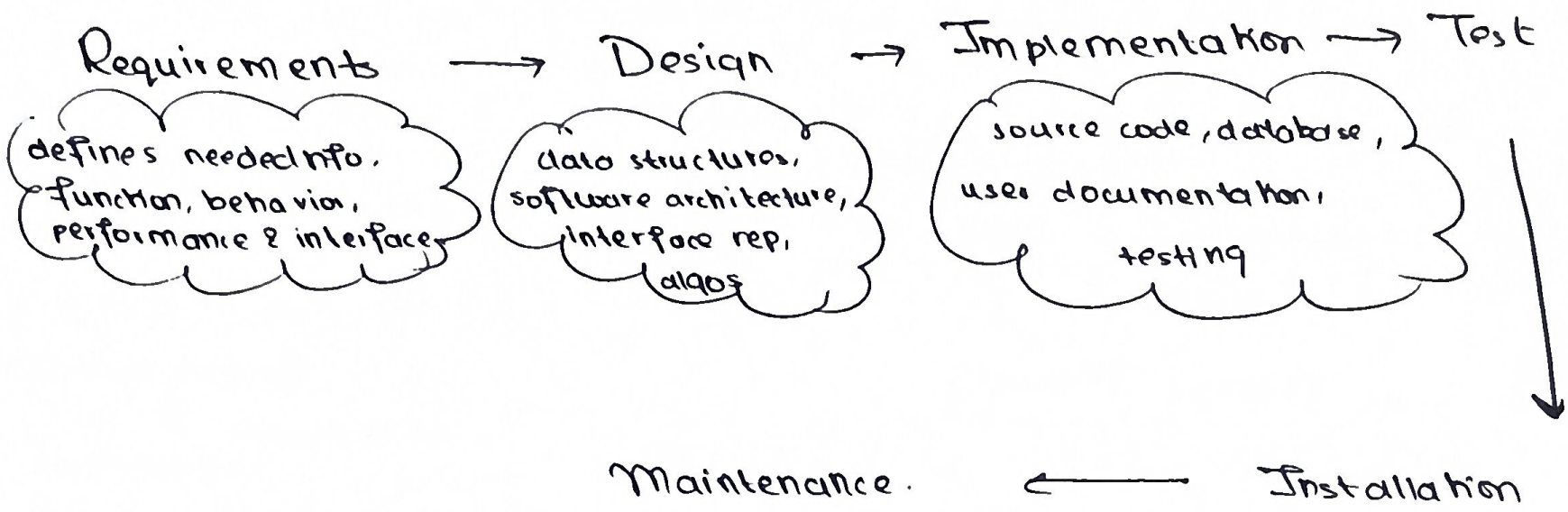
→ Formal methods
- provides a process and methodological process for defining aspects
- → AOSD
- → Unified process
  - ↳ use-case driven, architecture-centric, iterative & incremental a development objective.

when reuse is

mathematical specification of requirements

* Waterfall Model

→ Each step of the software engg. is followed sequentially.



Strengths

→ easy to understand & easy to use

→ provides structure to inexperienced staff

→ milestones are well understood.

→ sets requirements stability

→ good for management control

→ works when quality is more important than cost or schedule.

Deficiencies

→ All requirements must be known upfront

→ Deliverables created for each phase are considered frozen - inhibits flexibility

→ Can give a false impression of progress

→ Does not reflect the problem-solving nature of software development, no iterations

→ Integration is one big bang at the end

→ Little opportunity for customer to preview the system, until it may be too late.

When to use the waterfall model

→ Requirements are very well known

→ Product definition is stable

→ Technology is understood

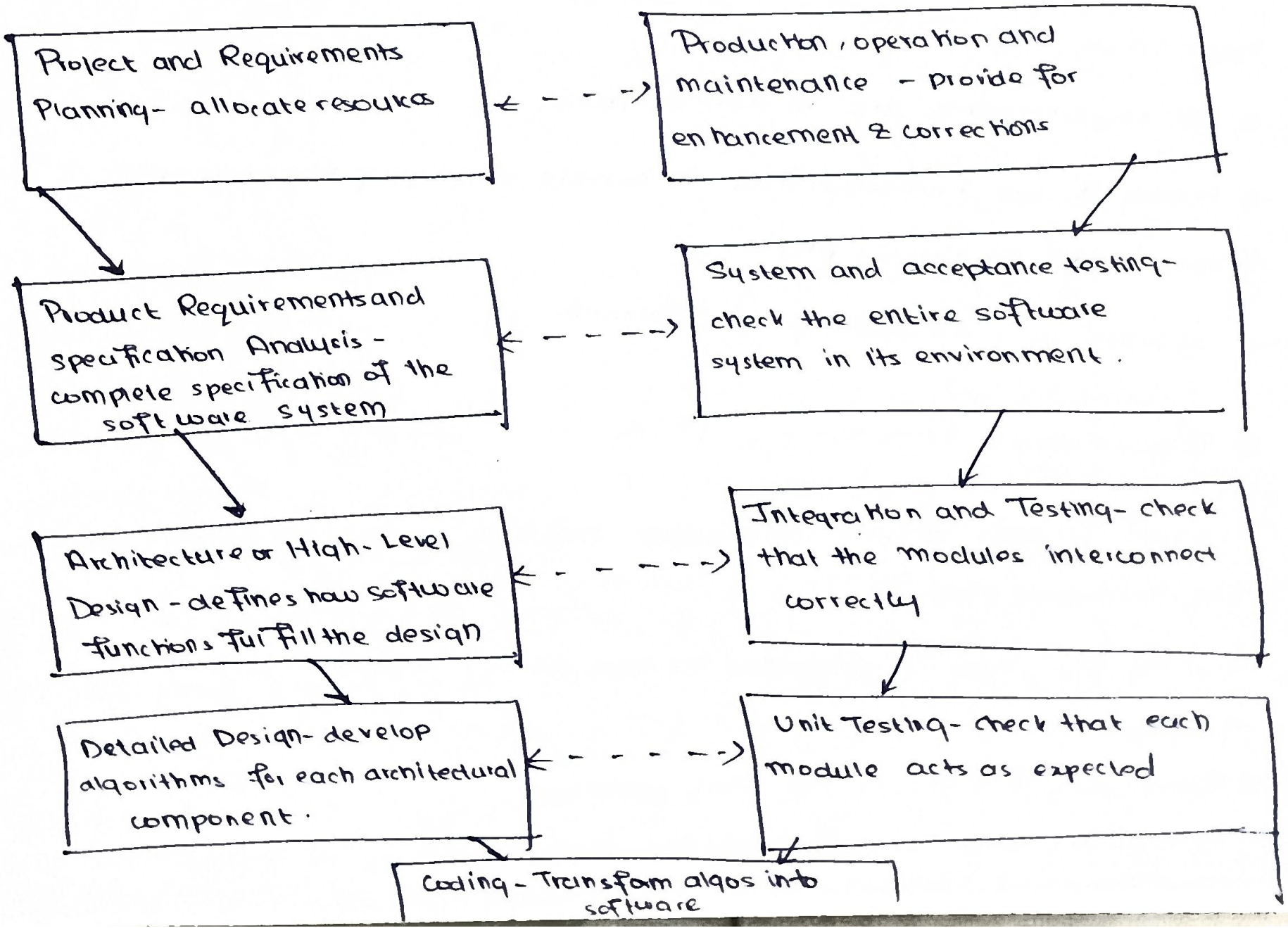
→ New version of an existing product

→ Porting an existing product to a new platform

* V-V model

→ A variant of the waterfall that emphasizes the verification & validation of the product

→ Testing of the product is planned in parallel with a corresponding phase of development.



Strengths

→ emphasizes planning for verification & validation in the early stages

→ Each deliverable must be testable

→ Project management can track progress using milestones

Weaknesses

→ Does not easily handle concurrent events

→ Does not handle iterations or phases

→ Does not easily handle dynamic changes in requirements

→ Does not contain risk analysis activities

When to use V-V model

→ when systems need high reliability - hospital patient control applications

→ All requirements are known upfront

→ When it can be modified to handle changing requirements beyond the analysis phase.

→ Solution and technology are known.

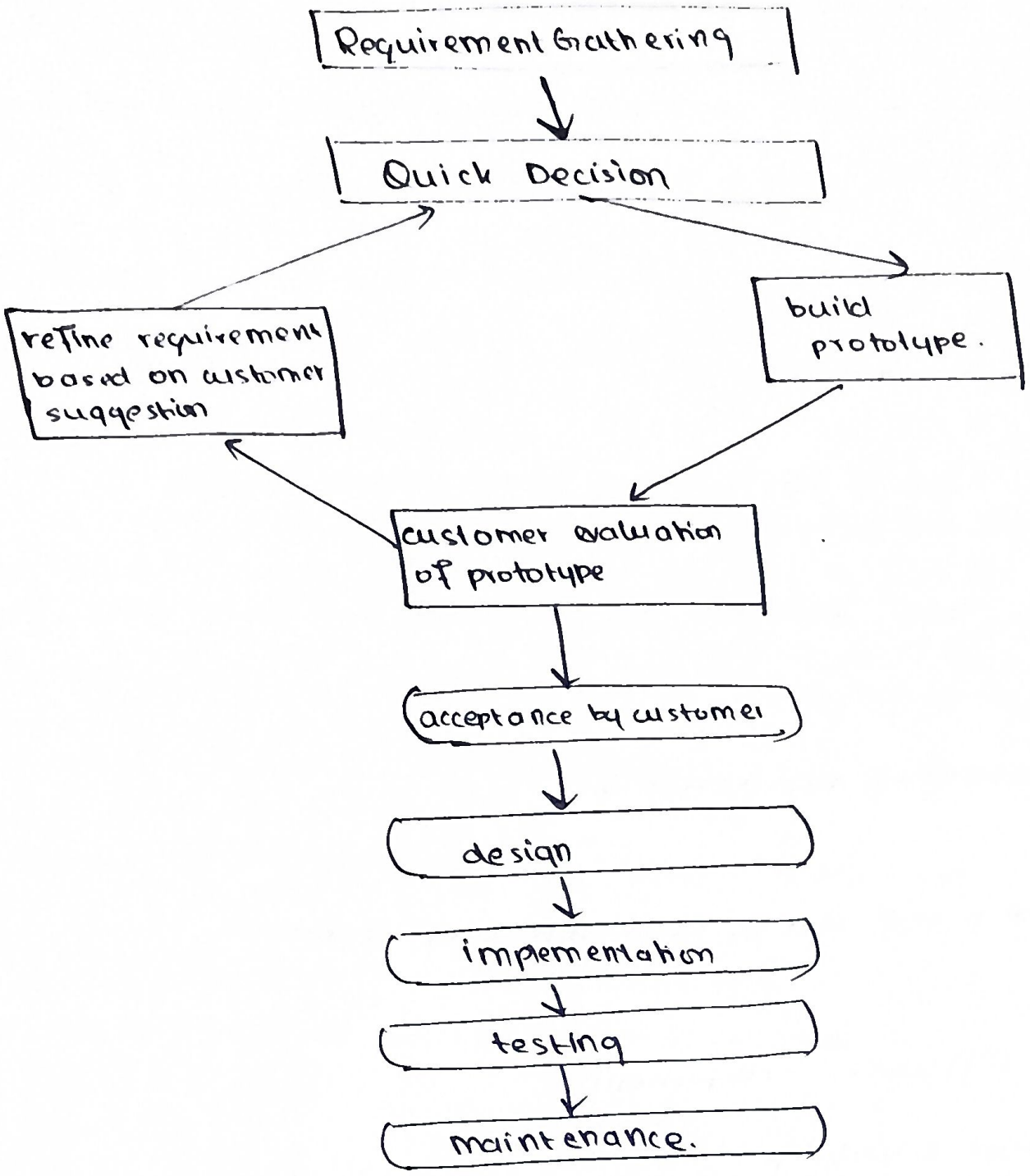
* Evolutionary Prototyping Model

→ In this model, developers build a prototype during the requirements phase. (w/ db, UI, algo)

→ The prototype is evaluated by end users, who give corrective feedback

→ Developers further refine the prototype.

→ When the user is satisfied, the prototype code is brought up to the standards needed for a final product.



Strengths

→ Customers can see system requirements as they are being gathered

→ unexpected requirements accommodated

→ more accurate end product

→ steady, visible signs of progress

→ devs learn from customers

Weaknesses

→ Tendency to abandon structured program

→ Bad reputation for quick and dirty methods

→ Maintainability may be overlooked

→ Process may continue forever (scope creep)

When to use Evolutionary prototyping

→ When requirements are unstable / have to be clarified

→ develop UI

→ short-lived demonstrations

→ new, original development

* RAD model- Rapid Application Development Model

→ Is a concept that products can be developed faster and of higher quality by:

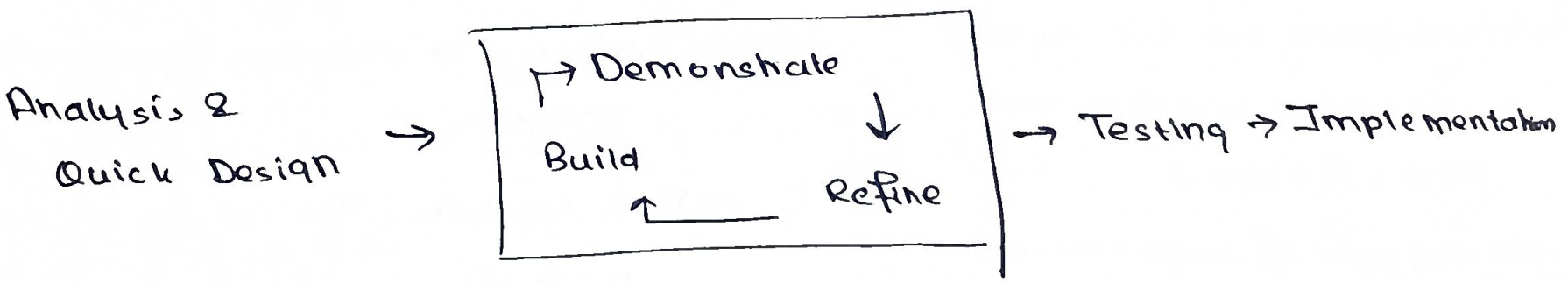
(i) gathering requirements

(ii) Prototyping and early, reiterative user testing

(iii) Reuse of software components

(iv) Rapidly paced schedule

(v) less formality in reviews



Strengths

→ Reduced cycle time & improved productivity

→ Time box approach mitigates & scheduled risk

→ minimizes risk of not achieving customer satisfaction

→ Focus moves from documentation to code

Weaknesses

→ must give quick responses to the user

→ Risk of never achieving closure

→ Hard to use with legacy systems

→ Requires a system that can be modularized

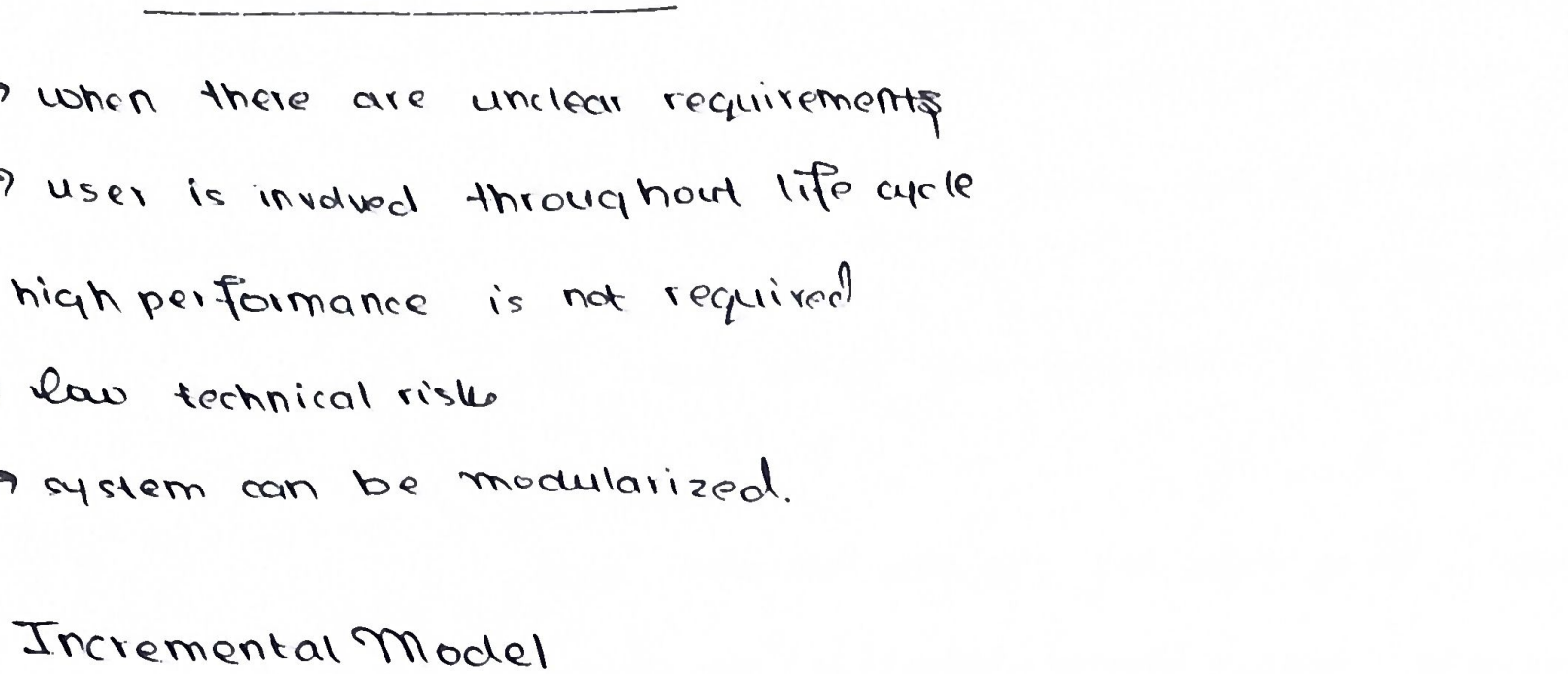
→ Dev & customers must be committed to rapid fire activities.

* When to use RAD

→ when there are unclear requirements
→ user is involved throughout life cycle
→ high performance is not required
→ low technical risks
→ system can be modularized.

* Incremental Model

→ Construct a partial implementation of a total system, and then slowly add increased functionality
→ Prioritize requirements of the system and implement them in groups
→ Each partial release adds function to the previous release.



Strengths

→ Develop high-risk or major fns. first
→ Each release delivers an operational product
→ Customer can respond to each build
→ Lower inital delivery cost, deliver product faster

Weaknesses

→ Requires good planning & design
→ define and complete fully func. system early on
→ must be modularized
→ Total cost of system is not not lower.

When to use the Incremental Model

→ when staffing is unavailable

→ risk, funding, schedule, program benefits or need for early realization of benefits

→ requirements are known up-front but are expected to evolve over time

→ get basic functionality to the market early

→ on projects which have lengthy development schedules

→ on products w/ a new technology

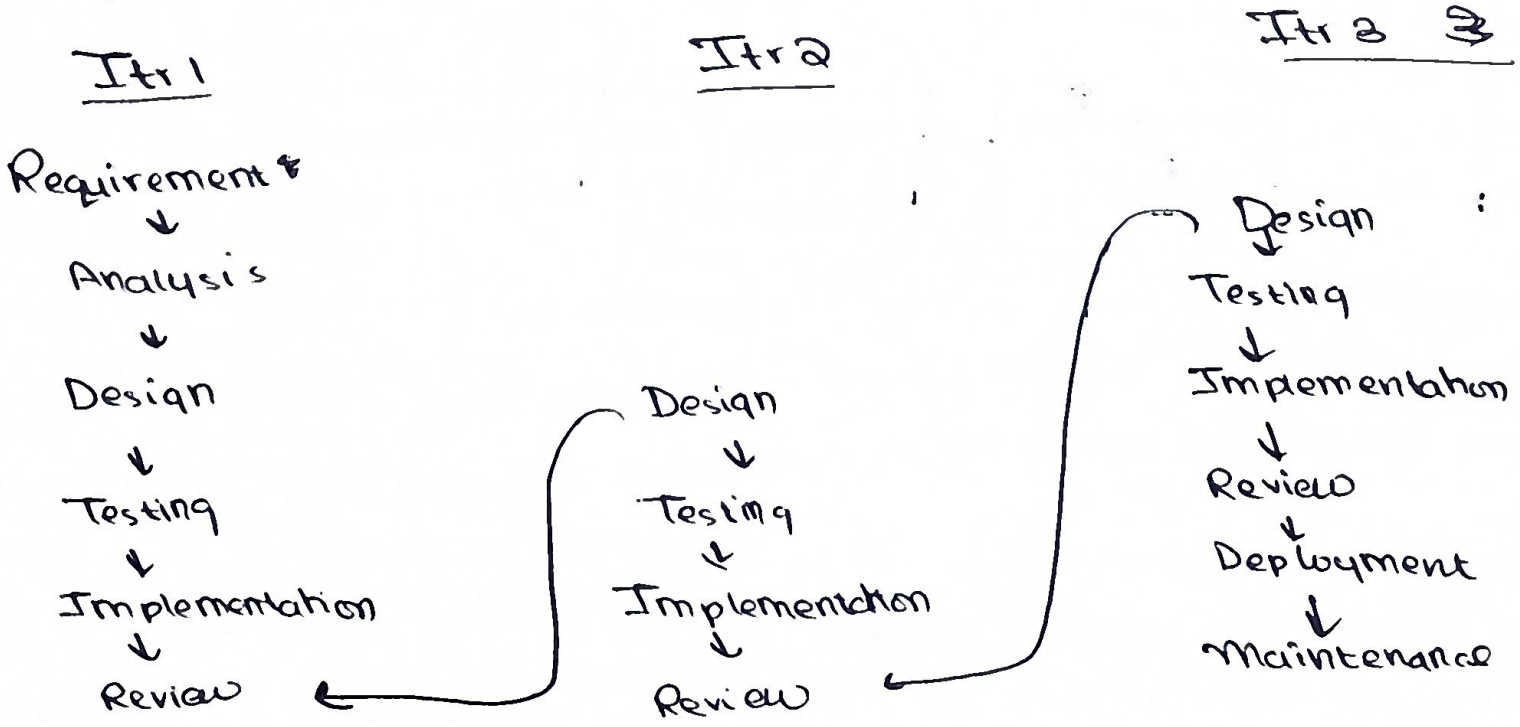
* Iterative Model

→ Begin development with some of the software specification and develop the first version of the software.

→ After the first version, if there is a need to change the software, then a new version of the software is created with a new iteration

→ Every release of the iterative model finishes in an exact and fixed period, that is called an iteration.

→ The iterative model allows accessing earlier phases, in which different variations are made.



Pros

→ easy testing and debugging

→ can have parallel development

→ acceptable to ever-changing needs

→ limited time on documentation and more time on designing

Cons

→ not suitable for smaller projects

→ more resources may be required

→ design can change again & again

→ changes can cause over budget

→ project completion date not confirmed.

When to use the Iterative Model

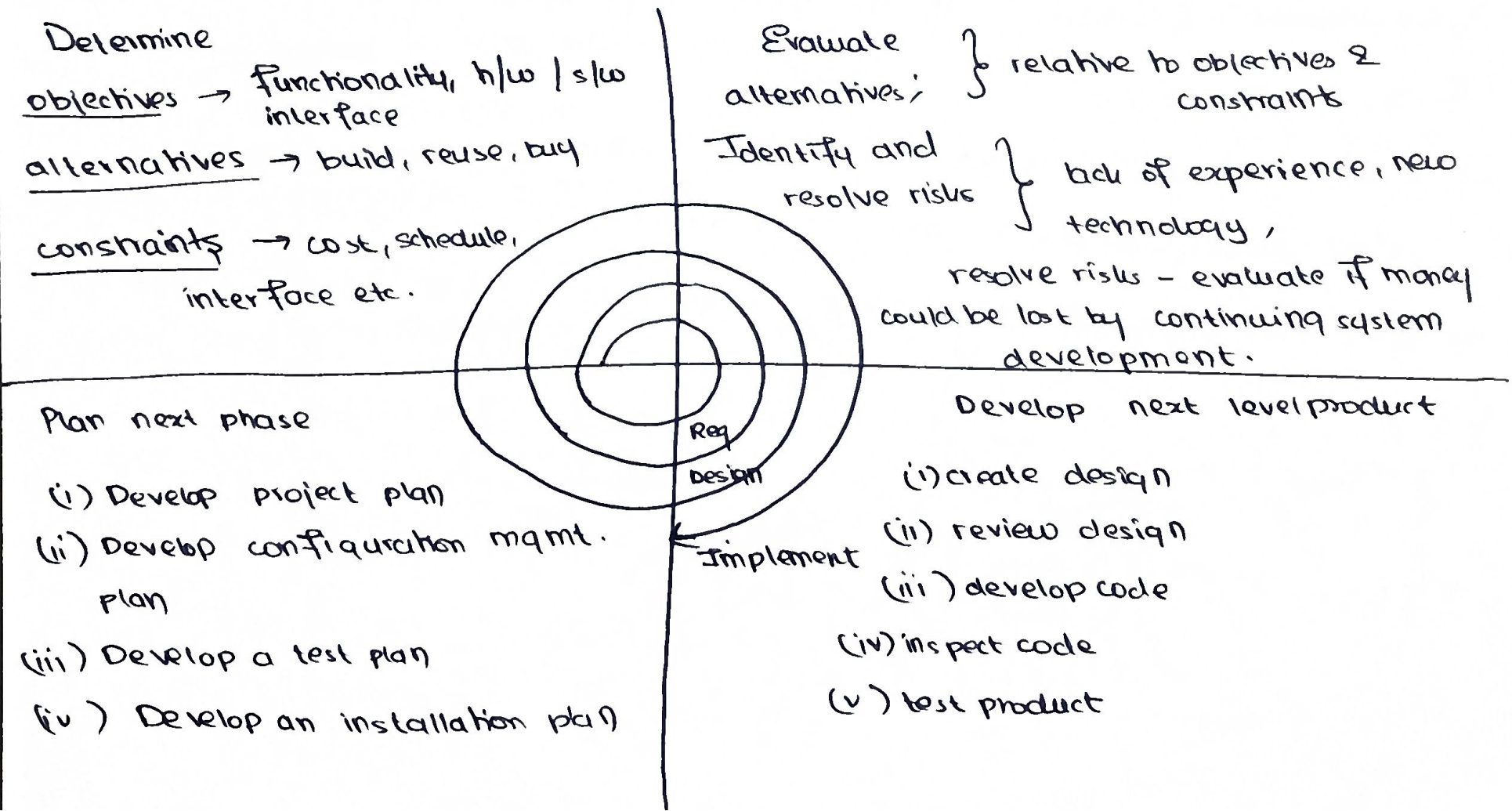
→ when the software application is large

→ when there is a requirement change in the future.

* Spiral Model

→ Adds risk analysis and 4gl RAD prototyping to the waterfall model

→ Each cycle involves the same sequence of steps as the waterfall process model.



Strengths

→ early indication of risks
→ users see system early because of rapid prototyping tools.
→ high risk functions are developed first
→ design does not have to be perfect
→ cumulative costs assessed frequently

Weaknesses

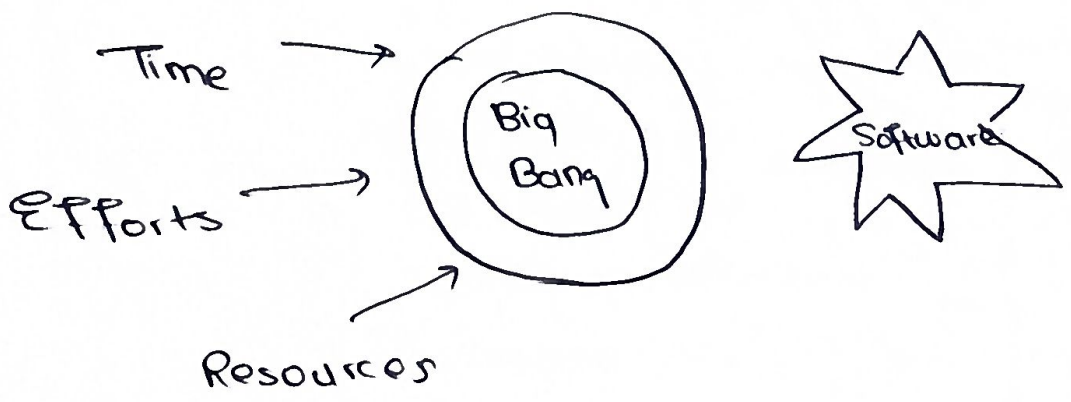
→ Too much time spent on evaluating risks
→ Excessive time spent planning, resetting objectives, doing risk analysis
→ Spiral may continue indefinitely
→ May be hard to define objectives, verifiable milestones

When to use Spiral Model

→ R & D / large-scale projects where privacy / security issues are involved
→ when costs and risk evaluation is important
→ users are unsure of needs
→ significant changes are needed.

* Big-Bang Model

→ developers do not follow any specific process
→ Development begins with the necessary funds and efforts in the form of inputs
→ Result may / may not be as per customer's requirements as customer's requirements are not defined.
→ For small projects like academic projects. 1/2 devs can work on this



Pros	Cons
→ no planning	→ high risk and uncertain
→ simple model	→ not acceptable for a large project
→ few resources needed	→ if requirements not clear, then it can be very expensive.
→ flexible for developers	

When to use Big Bang Model

→ project is small (academic)

→ size of developer team is small

→ no defined requirements

→ release date not confirmed.

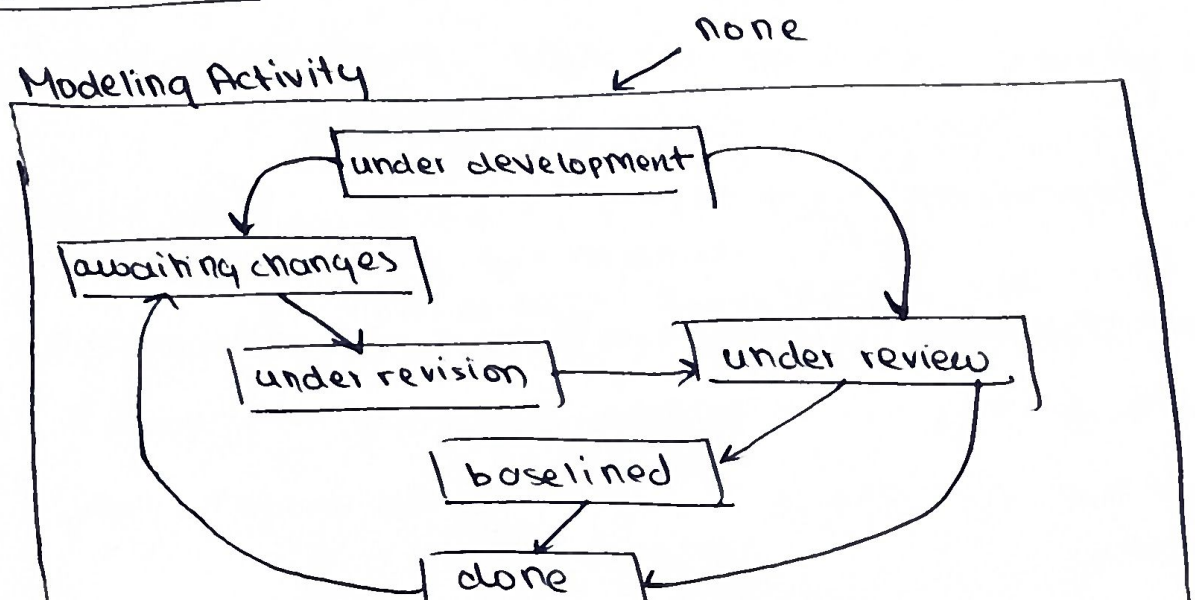
* Evolutionary Models: Concurrent

→ a series of framework activities, s/w engg. tasks and their states

→ Early on in a project, when the customer communication activity is completed, the project is in the awaiting changes states.

→ The modeling activity which existed in the none state while inital customer communication was completed makes a transition to the under development state.

→ If the customer indicates a change, the activity moves from the under development stage to the awaiting changes state.



* Agile SDLCs.

→ Agile software development refers to a group of software development methodologies based on iterative development.

Phases

1. Requirements gathering
2. Design the requirements
3. Construction / iteration
4. Testing / Quality Assurance
5. Deployment
6. Feedback

Characteristics

→ speed up or bypass one or more life cycle phases
→ usually less formal & reduced scope
→ used for time-critical applications
→ used in organizations that employ disciplined methods

Some Agile Methods

→ Scrum — based on team-based conditions, has scrum master, product owner scrum team
→ Adaptive Software Development (ASD)
→ Crystal Clear — chartering, cyclic delivery, wrap up
→ Dynamic Software Development Method (DSDM) - a RAD for s/w development and gives an agile project structure.

Pros

1. Frequent Delivery
2. Face-to-face communication
3. Reduce total dev time
4. Anytime changes are acceptable.

Cons

1. shortage of formal docs
2. confusion & misinterpretation
3. maintenance difficult h a lack of documentation

When to use Agile Model

→ Frequent changes are required

→ When the project is small

→ Highly qualified and experienced team

* Quality Assurance Plan

→ plan for quality assurance should be in writing

→ Decide if a separate group should perform the quality assurance activities

→ Plan should have :
- (i) defect-tracking → tracks each defect and if resolved
- (ii) unit testing → test each module
- (iii) source-code tracking → step through code line by line
- (iv) integration & system testing → exercise new code in combination w/ code that has already been integrated

system-testing - execution of the software to find defects.

* Software Process

→ A roadmap of predictable step, that helps one create a timely high-quality product or system

People Involved -
- → software engineers + managers
- → people who have requested the software
- → those who define, building and test it

Importance → provides stability, control & organization

→ if unconholled ⇒ chaos

→ a modern s/w engg. approach must be agile

→ should do only those activities, controls & work products that are appropriate for the project team.

Steps Involved → depends on the software you're building (avionics vs. website creation)

Work product → includes (i) programs & documents

(ii) data produced as a consequence of the activities and tasks.

Assessment Mechanism : (i) quality

(ii) timelines

(iii) long-term viability

(iv) efficacy

* Generic Process Model

→ A generic process framework for software engineering defines

Five framework activities

a. communication

b. planning

c. modelling

d. construction

e. deployment

Define a framework activity

Identify a task set

Process patterns

* Process Assessment and Improvement

A. Standard CMMI Assessment Method for Process Improvement (SCAMPI)

→ provides a 5 step process assessment model

(i) initiating
(ii) diagnosing
(iii) establishing
(iv) acting
(v) learning

B. CMM-Based Appraisal for Internal Process Improvement (CBA IPI)

→ provides a diagnostic technique for assessing the relative maturity of a s/w organization

C. SPICE - The SPICE (ISO/IEC 15504) standard

→ defines requirements for software process assessment

→ assist organizations in developing an objective evaluation of the efficacy of any defined software process.

D. ISO 9001 : 2000 for Software

→ a generic standard that applies to any organization that want to improve the overall quality of the products, systems or services it provides

* Software Development Paradigms

→ Paradigms consist of a set of assumptions, concepts, values and practices that constitutes a way of viewing reality for the community that shares them.

→ Some software development paradigms include:

(i) procedural paradigm

(ii) data - driven paradigm

(iii) object - oriented paradigm

(iv) ad - hoc paradigm

Procedural paradigms - → list the steps needed to solve the problem and then successively decompose those steps into smaller and more simpler sub-problems, which are represented with procedures, functions or methods.

→ focuses on HOW to solve a problem

problem → analysis and design → implementation

Data-driven paradigms

→ the analysis and design phases are logically close to the real world problem

→ Data driven analysis helps developers gain domain understanding.

→ The resulting data-flow graphs are easier for non-software developers to under However, the data flow diagrams are more difficult to convert to running programs

Object-oriented Paradigm.

→ encapsulate data and procedures or operations that may access and operate on that data into an entity called an object.

→ This paradigm evenly spreads the development phases w/o any large gaps.

→ Developers identify classes during analysis, refine them during design,

and implement them during the programming phase

→ The consistent use of classes forms the object-oriented bridges between each phase of software development.

* Software Development Practices

→ There are 7 top software development best practices to follow:

1. Create an SRDS
2. Keep the code simple
3. Have proper documentation
4. Focus on agile development
5. Always remember unit testing
6. Consider software maintenance beforehand
7. Conduct a thorough code review

(1) Create an SRDS

→ A software requirements and design specification document clearly defines the requirement of the project

→ It simplifies in identifying what resources to align to which tasks

→ includes software design characteristics, design diagrams, decisions constraints etc.

→ allows developers to figure out what the project demands w/o any confusion

(2) Keep the Code Simple

→ useful in reducing the complexity of the project

2 principles:

DRY - Don't Repeat Yourself

YAGNI - You Ain't Gonna Need It

→ simplified code ⇒ high-quality product.

(3) Have Proper Documentation

→ useful for 2 reasons:

(i) why there is a need to write that code

(ii) keeps everyone on the same page

→ devs can collaborate easily.

(4) Focus on Agile Development

→ helps the dev team to better understand the product. Offers quick iterations that lead to faster product development. Improvements are quick and the product is of high-quality

→ Offers great collaboration between teams. Fast moving pieces enable everyone to focus on achieving the same goal.

(5) Always Remember Unit Testing

→ ensures that code is free of bugs & vulnerabilities

→ Unit testing assists with individual ~~to~~ functions and classes

→ helps documenting code

(6) Consider Software Maintenance Beforehand

→ Enterprise must continuously improve the product, reduce deterioration, satisfy users and ensure continued success of the software.

→ Must keep the product functional and healthy.

(7) Conduct a Thorough Code Review

→ Reviewing ensures that the code is of high-quality. Senior developers and engineering experts know what goes into excellent code

→ No error is left behind

→ Provide objective opinion on improvement.

* Project Planning Process

1. Define Stakeholders → Stakeholders include anyone with an interest in the project. They can include the customer or end user, members of the project teams etc

2. Define roles → Each stakeholder's role must be clearly defined. Some people may fill multiple roles.

3. Introduce stakeholders → Hold a meeting to bring stakeholders together and unify the vision behind the project. The topics covered should include scope, goals, budget etc

4. Set goals → Take what is learned from the meeting and refine it into a project plan. It should include goals and deliverables that define what the product or service will result in.

5. Prioritize tasks → List necessary tasks and prioritize them based on importance and interdependencies, use a Gantt chart

6. Create a schedule → Establish a timeline that considers the resources needed for all the tasks.

7. Assess risks : Identify project risks and develop strategies for mitigating them.

8. Communicate Share the plan with all stakeholders and provide communication updates

9. Reassess : As milestones are met, revisit the project plan, and revise any areas that are not meeting expectations

10. Final evaluations: Once the project is completed, performance should be evaluated to learn from the experience and identify areas to improve.

* Software Project Management

* The 4 P's

→ (i) People - the most important element of a successful project

(ii) Product - the software to be built

(iii) Process - the set of framework activities and software engg. tasks to get the work done

(iv) Project - all the work required to make the product a reality

* Stakeholders

(i) Senior Managers → define business issues that have influence on the project

(ii) Project (Technical) Managers → plan, motivate, organize and control practitioners who do software work.

- (iii) Practitioners - deliver the technical skill necessary

(iv) Customers - specify software requirements

(v) End-users - those who interact with the software.

Software Engineering

Numericals

Type1 - Conventional Methods

A. LOC Approach

expected value $S = \frac{S_{opt} + 4s_m + S_{pess}}{6}$ ← most likely

$$\text{Cost of LOC} = \frac{\text{average labor cost}}{\text{average productivity}}$$

$$\text{project cost} = \text{cost of LOC} \times \text{total lines of code}$$

$$\text{project effort} = \text{total lines of code} / \text{productivity}$$

Example For the given fns, and the no. of lines of code specified, find the expected value, the cost of LOC, project cost & project effort given that

optimistic estimation = 4700

most likely estimation = 6000

pessimistic estimation = 10,000

Average productivity is 500 LOC/pm

average labor cost is $6000 pm

	LOC
F1	2500
F2	5600
F3	6450
F4	3100
F5	4740
F6	2250
F5	7980

Ans Total lines of code = 32,620

$$S = \frac{S_{opt} + 4S_m + S_{poss}}{6}$$

$$= \frac{4700 + (4 \times 6000) + 10{,}000}{6}$$

$$S = 6450$$

$$\text{Cost of LOC} = \frac{\text{Total project cost}}{\text{avg. productivity}} = \frac{6000}{500} = 12$$

Total product cost = cost of LOC x total lines of code

$$= 32{,}620 \times 12$$

$$= ₹391440$$

$$\text{Total estimated project effort} = \frac{\text{Total LOC}}{\text{productivity}}$$

$$= \frac{32620}{500} = 65 \text{ person months}$$

B. FP-based estimation

Example 1 - Consider the following FP components and their complexity. If the total degree of influence is 52, find the estimated function points

Function Type	Estimated Count	Complexity
ELF	2	7
ILF	4	10
EQ	22	4
EO	16	5
EI	24	4

$$FP = UFC \times VAF$$

$$UFC = \Sigma(\text{estimated count} \times \text{complexity})$$

$$VAF = 0.65 + (0.01 \times \Sigma(Fi))$$

here, $UFC = 14 + 40 + 88 + 80 + 96$

$$UFC = 318$$

$$VAF = 0.65 + (0.01 \times 52)$$

$$VAF = 1.17$$

$$FP = UFC \times VAF = 318 \times 1.17$$

$$= 372.06$$

$$FP \simeq 372$$

Example Given the function point data:

i. A target product has 8 simple inputs, 4 average inputs and 12 complex inputs.

ii. There are 66 average outputs, 18 simple inquiries, 10 avg. master files and 7 complex interfaces.

iii. The total degree of influence is 46. Estimate the no. of fn points

Component	Level of Complexity		
	simple	average	complex
External i/ps	3	4	6
External o/ps	4	5	7
External inquiries	3	4	6
Internal logical files masters	7	10	15
External interface files	5	7	10

$UFC_1 = (8 \times 3) + (4 \times 4) + (12 \times 6)$

$= 112$

$UFC_2 = (66 \times 5) + (18 \times 3) + (10 \times 10) + (7 \times 10)$

$= 554$

$FP = (UFC_1 \times VAF) + (UFC_2 \times VAF)$

$= (UFC_1 + UFC_2) \times VAF$

$VAF = 0.65 + [0.01 \times \Sigma F_i] = 1.11$

$\Rightarrow FP = (554 + 112) \times 1.11$

$FP = 739.26$

Type 3 - Basic Cocomo Model

Example : Given a KLOC of 33.2, find the estimation for the product

	a_b	b_b	c_b	d_b
organic	2.4	1.05	2.5	0.38
semi detached	3.0	1.12	2.5	0.35
embedded	3.6	1.20	2.5	0.32

organic

$$E = a_b(KLOC)^{b_b}$$

$$E = (2.4)\ (33.2)^{1.05}$$

$$E = 94.92 \text{ pm}$$

$$D = c_b(KLOC)^{d_b}$$

$$D = 2.5 (94.92)^{0.38}$$

$$= 2.5 \times 5.614$$

$$D = 14.103 \text{ month}$$

$$P = E/D = 94.92 / 14.03$$

$$P = 6.731 \quad \rightarrow (1)$$

Illy do for semi-detached & embedded

semi-detached $E = 151.63$ $D = 4.4925$ $P = 10.468$ → (2)

embedded $E = 240.804$ $D = 14.456$ $P = 16.65$ → (3)

The best model is the organic one with the least P & E.

Intermediate COCOMO:- $a_i(KLOC)^{b_i} \times EAF$ person months

Type 4 - COCOMO II

$$NOP = OP * (100 - \%\ reuse)/100$$

$$effort = \frac{NOP}{person\text{-}month}$$

Type 5 - Earned value analysis

① Budgeted cost = Rs 9,00,000

project is to be completed in 9 months

After a month, 10% of the project has been completed at a total expense of Rs 1,00,000.

The planned completion should have 15%.

Determine whether the project is on time & on budget using EVA.

Ans Budgeted ~~actual~~ at cost (BAC) = Rs 9,00,000.

AC = Rs 1,00,000

planned value $PV = \frac{15}{100} \times 9,00,000$

$$PV = 1,35,00$$

expected value $EV = \frac{10}{100} \times 9,00,000$

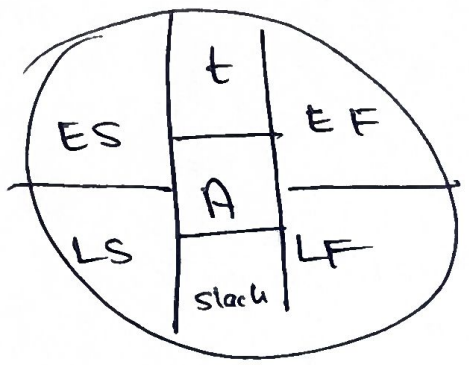
$$Ev = 90,000$$

$$CPI = \frac{EV}{AC}$$

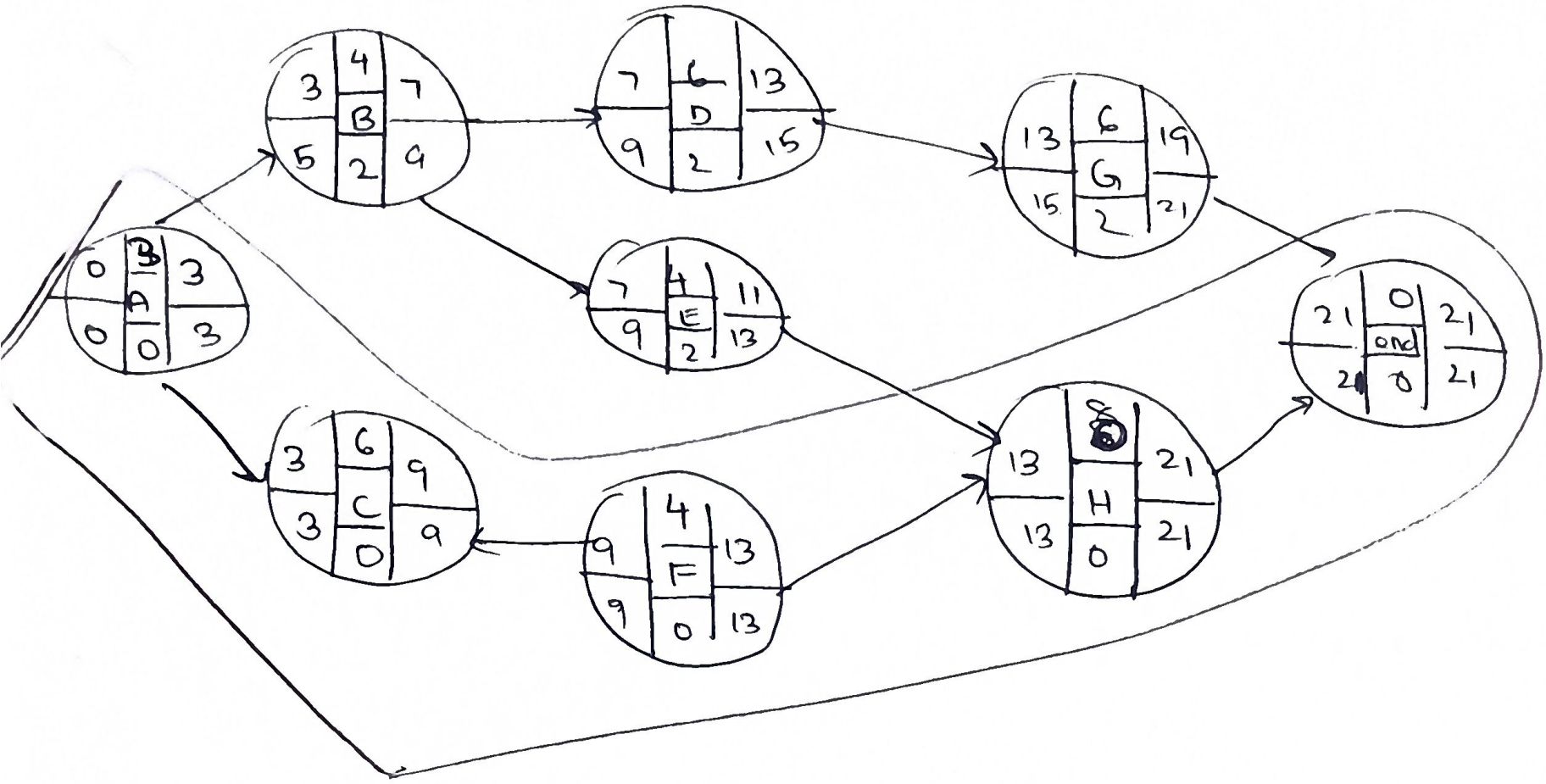
$$= \frac{90,000}{1,00,000} = 0.9 \text{ //}$$

$$SPI = \frac{EV}{PV} = \frac{90,000}{1,35,000} = 0.67 \text{ //}$$

Type6 CPM



Activity	Predeccessa	Time
A	—	3
B	A	4
C	A	6
D	B	6
E	B	4
I	C	4
G	D	6
H	E, F	8


3 4 7
B
5 2 9
7 6 13
D
9 2 15
13 6 19
G
15 2 21
0 B 3
A
0 0 3
7 4 11
E
9 2 13
21 0 21
END
21 0 21
3 6 9
C
3 0 9
9 4 13
F
9 0 13
13 8 21
H
13 0 21