PERT (Contd.)

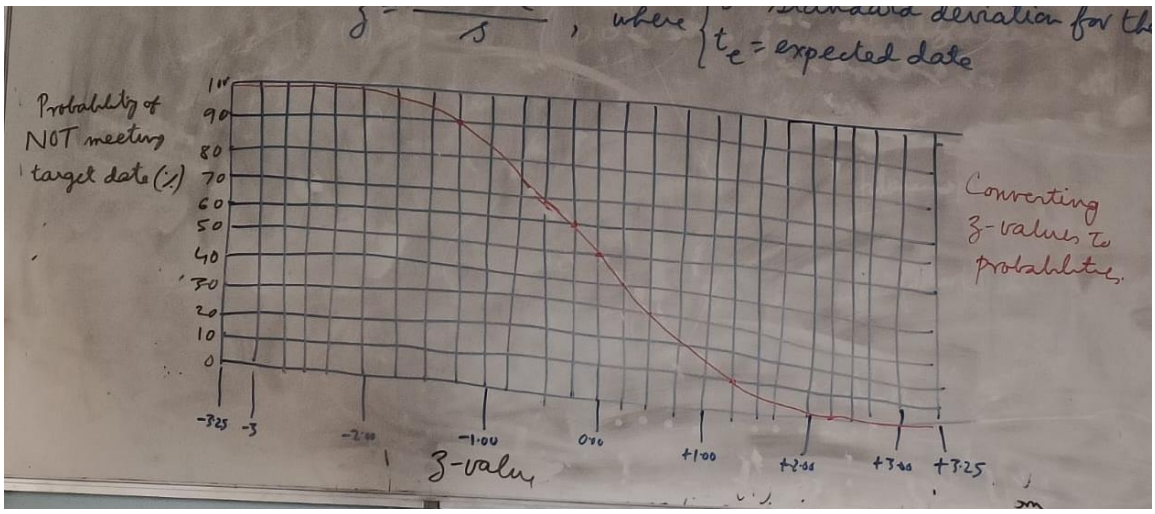
Three-step method for calculating the probability of meeting or missing a target date

i. Calculate the stabndard deviation of each project event
ii. Calculate the z-value for each event that has a target date;
iii. Convert z values to probabilities

Calculating Z values

For a node having a target date T,

$$z = \frac{T - t_e}{s}, where \begin{cases} s = Standard\ deviation\ for\ the\ event \\ t_e = expected\ date \end{cases}$$



Earned value analysis

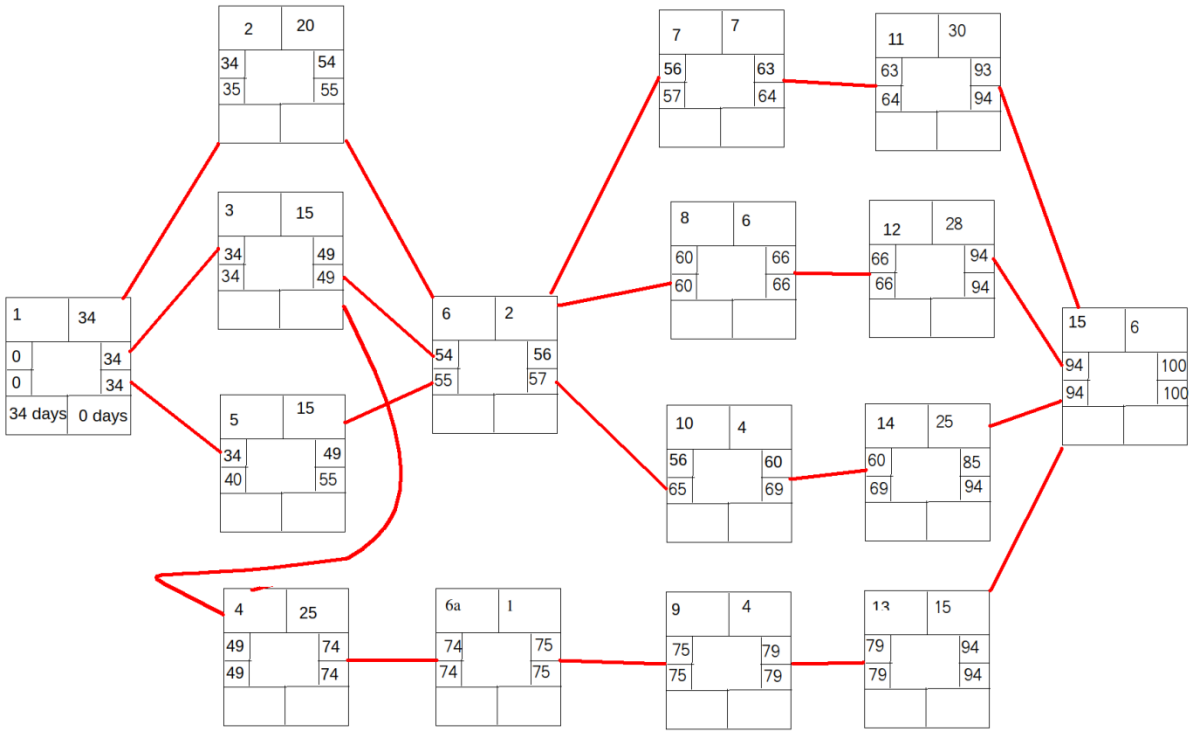
Objective: Monitoring progress

Earned value analysis is based on assigning a 'value' to each task based on the original expenditure forecasts.

BCWS (budgeted cost of work scheduled) is the original budgeted cost of the item. [also called the "baseline budget"]

BCWP (budget cost of work performed) is the total value credited to a project at any point [also called "earned value"]

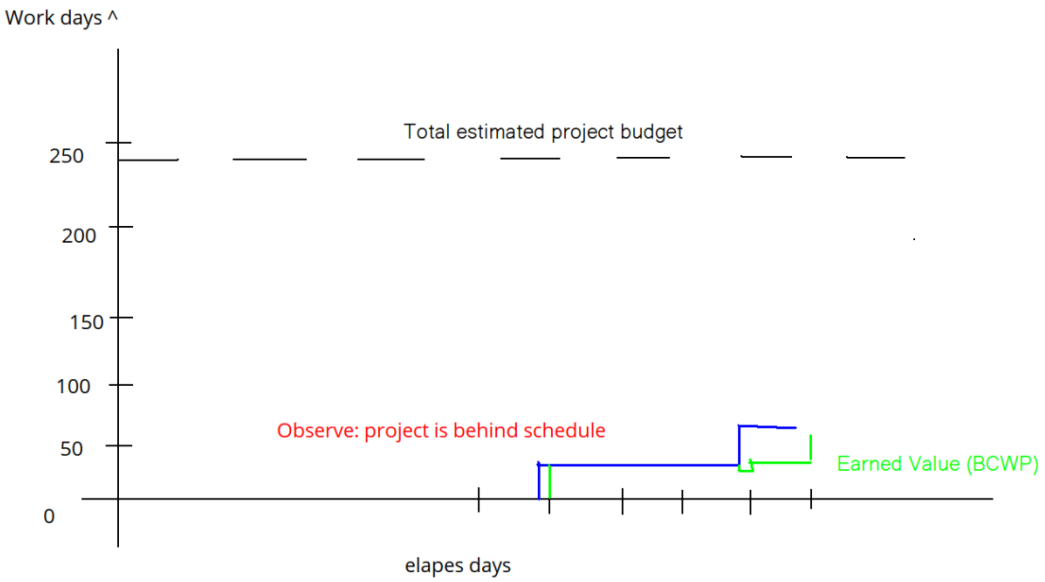
TASK	Budgeted work days	Scheduled completion	Cumulative work-days	% cumulative earned value
1	— 34	— 34	— 34	— 14.35
3	— 15	— 49	— 64	— 27.00
5	— 15	— 49		
2	— 20	— 54	— 84	— 35.44
6	— 2	— 56	— 86	— 36.28
10	— 4	— 60	— 90	— 37.97
7	— 7	— 63	— 97	— 40.93
8	— 6	— 66	— 103	— 43.46
4	— 25	— 74	— 128	—
6a	— 1	— 75	— 129	— 54.43
9	— 4	— 79	— 133	— 56.12
14	— 25	— 85	— 158	— 66.67
11	— 30	— 93	— 188	— 79.32
12	— 28	— 94	— 231	— 97.47
13	— 15	— 94		
15	— 6	— 100	— 237	— 100



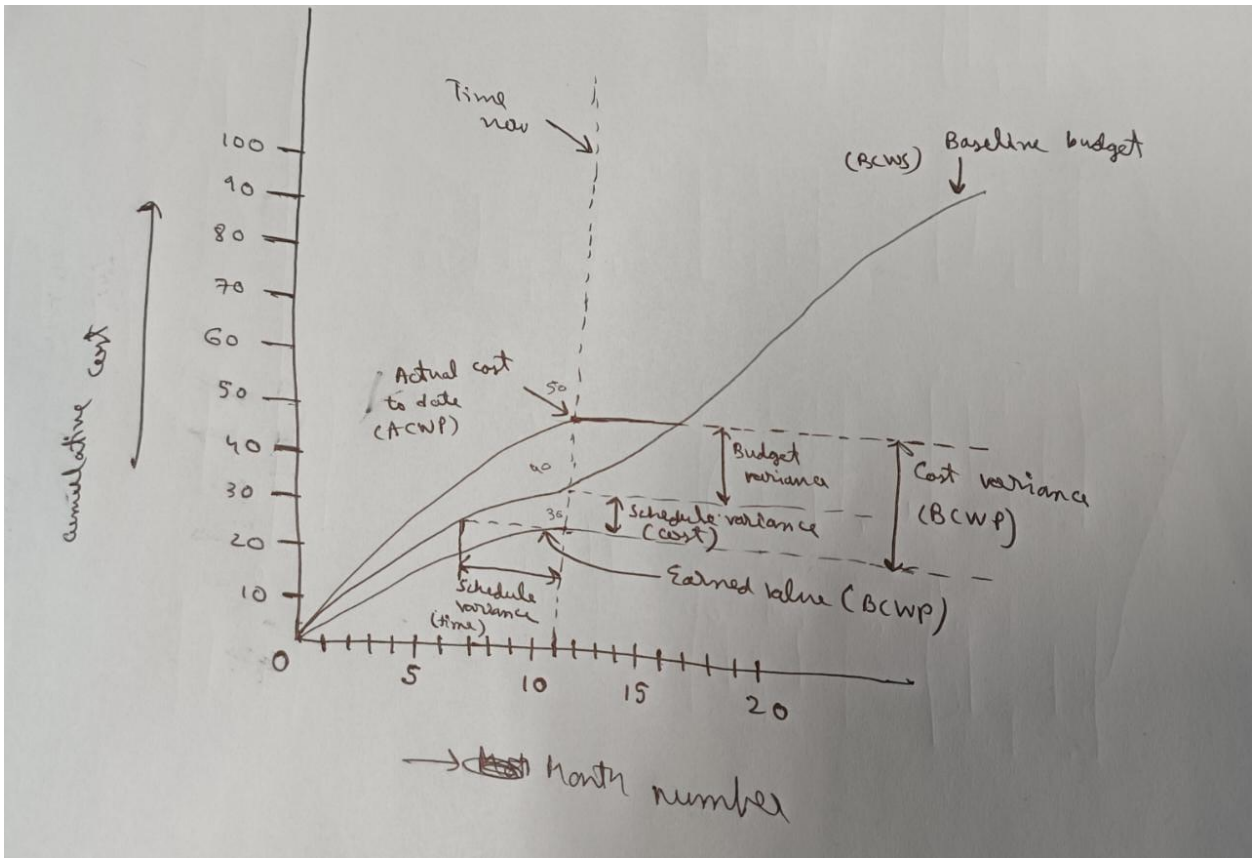
Cost Varience = BCWP – ACWP

Schedule Varience = BCWP – BCWS

$$Earned\ Value = \frac{BCWP}{\sum BCWS}$$

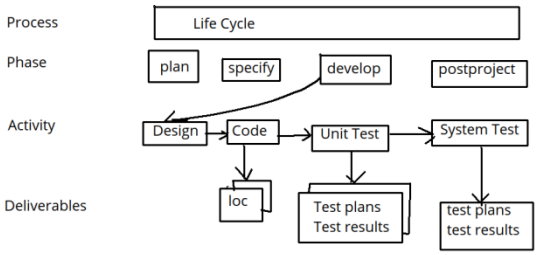


MONITORING EARNED VALUE



Software Life cycle Model (SLCM)

SLCM graphicall describes how software development activites will be performd by depicting the sequence of activities.



Waterfall model

1. Feasibility study ↴
   2. Requirement analysis and specification ↴
      3. Design ↴
         4. Coding and unit testing ↴
            5. Integration and system testing ↴
               6. Maintenance

1. **Feasibility Study:** to determine whether it would be financially and technically feasible to develop the software.
   a. **Actions:**
      i. develop an overall understanding
      ii. Formulate various possible strategies
      iii. Evaluate different solutions and strategies
2. **Requirement Analysis and design**
   a. Gather requirements from customer -> Analyse (to remove inconsistencies and incompleteness
   b. Prepare SRS (Software requirements specification document)
3. **Design:**
   a. **Goal:** to transform the requirements specified in the SRS document into a structure that is suitable for implementation in some programming language. In other words, the Software architecture is derived from the SRS document
4. **Coding and unit testing**
   a. **Purpose:** to translate a software design into source code AND to ensure that individually each function is working correctly
   b. **End product of this phase:** A set of program modules that have been individually unit tested
   c. **Unit testing:** activities include designing test cases, testing, debugging to fix problems and management of the test cases
5. **Integration and System testing**
   Integration of various modules are normally carried out incrementally over a number of steps
   Objective: To verify that the interfaces among various 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 SRS)
6. **Maintenance**
   a. **Maintenance:** 60% effect
   b. **Development**: 40% effect
   - **Corrective maintenance:** to correct errors that were not discovered during the product development phase

- **Perfective maintenance:** to improve the performance of the system, or to enhance the functionalities of system based on curtomers request.
- **Adaptive maintenance:** For porting the software to work in a new environment

SHORTCOMINGS of the classical waterfall model

i. **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. **It is 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. Customers requirements usually keep 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. **No overlapping of phases:** For efficient utilization of manpower, phases should overlap. For example, design test cases can overlap with design.

PROTOTYPING Model

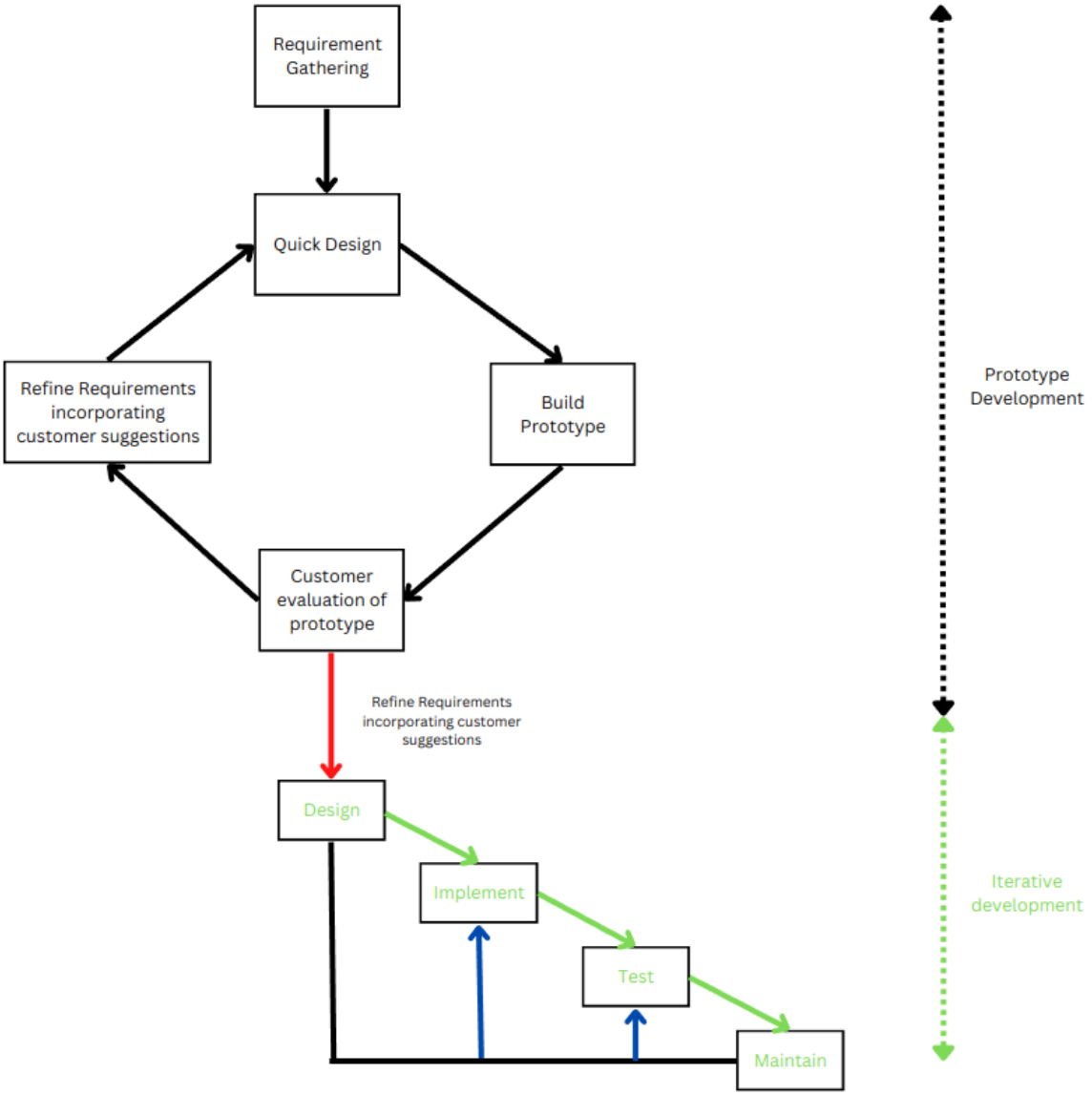
Approach: Build a working PROTOTYPE **Before** the development of the actual software.

Applicability:

i. Development of GUI (graphical user interface): 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 the development of highly optimized and efficient software is required

Two Major Activities
A. Prototype construction (it is thrown away later)
B. Iterative waterfall based software development (SRS document is also developed)

Weakness: It is effective only for those project for which the risks can be identified upfront before the project starts



Rapid Application Development (RAD)

- Code reuse is advocated. Thus to use RAD, a company should have had developed similar software earlier
- It is NOT suitable for cased where optimum reliability (e.g. Operating system) or performance (e.g. flight simulator) is required
- It is suitable customized software (developed for one or two customers by adopting an existing software)
- It is suitable for project with very aggressive time schedule

Working of RAD

RAD takes place in a series of short cycles of iteration. Each iteration is planned to enhance the implemented functionality of the application by only a small amount

[A prototype is developed for the functionality. The customer evaluated it an gives his feedback. The prototype is then developed]
It contrast, the PROTOTYPE life cycle model throws away the prototype

Other characteristics of RAD.

- The user is involved in **ALL phases of the life cycle** not only requirements definition, but design development test, and final delivery as well
- Use of product tools, e.g. code generators, screen generators, required during construction phase

Weakness of the RAD MODEL

W1. If the users can't be involved consistently throughout the life cycle, the final product the life cycle, the final product will be adversely affected

W2. Developers must be well-trained in the use of development tools

W3. It can fail if reusable components are not available

Strength if RAD Model

S1. The cycle time for the full product can be reduced due to the use of powerful development tools.

S2. Fewer developers are reqired because the system is developed by a project team familiar with the problem domain

S3. Quick initial views of the product are possible

S4. Ongoing customer involvement minimized the risk of not achieving customer satisfaction

When to use RAD (Applicability)

A1. On systems that may be modularized (component- based construction) and that are scalable

A2. On systems with reasonably well known requirements

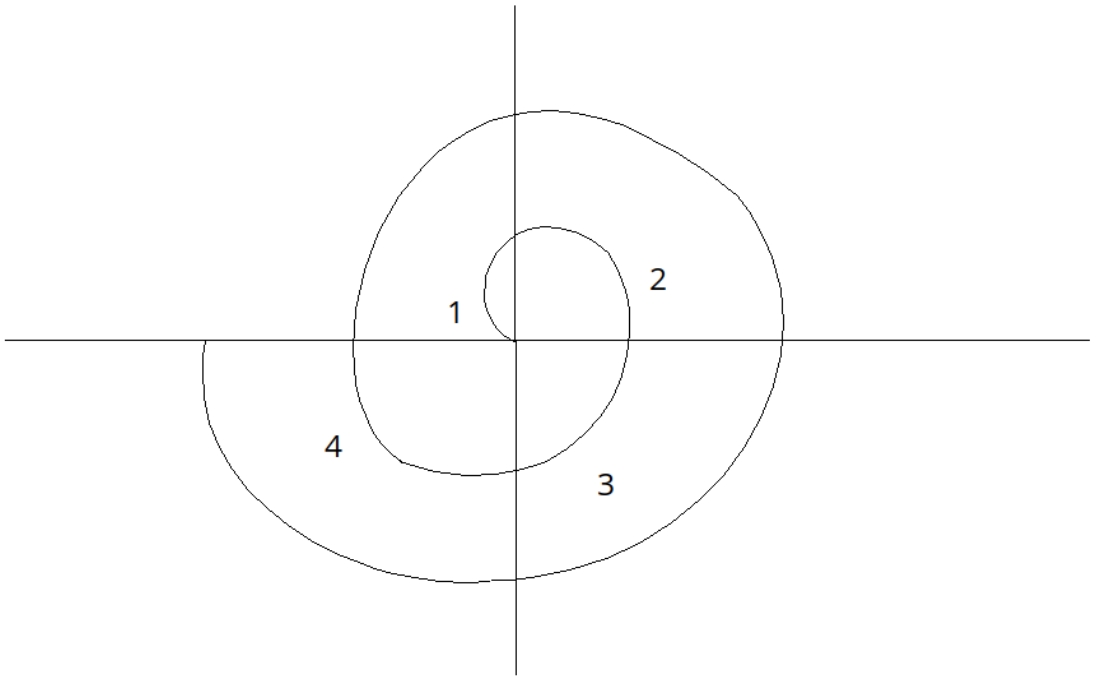
A3. When the end user can be involved throughout the life cycle

A4. When users are willing to become heavily involved in the use of automated tools

A5. When the technical risks are low

A6. When the project team is familiar with the problem domain and skilled in the use of development tools

A7. On system that can be "time-boxed" to deliver functionality in increments



<u>Quadrant 1</u>: Determine Objectives **(Performance functionality ability to accommodate changes hardware/software interface critical success factors)** alternatives **(Build reuse buy subcontract),** and constraints (**Cost schedule interface environmental limitations)**

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

<u>Quadrant 2:</u> Evaluate alternatives, and identify and resolve risks

Solutions are evaluated by developing an appropriate prototype

<u>Quadrant 3</u>: develop next level product

Creating of Design, review of design, development of code, and inspection of code, testing and packaging of the prduct. At the end of this quadrant, the identified features have been implemented and the next version of the software is available**.**

<u>Quadrant 4</u>:

- Development of project plan
- Development of Configuration management plan
- Development of test plan
- Development of installation plan

To sum up: review the developed version of the software with the customer and the plan of next iteration of the spiral

<Class 6>

Spiral Model (Cotd.)

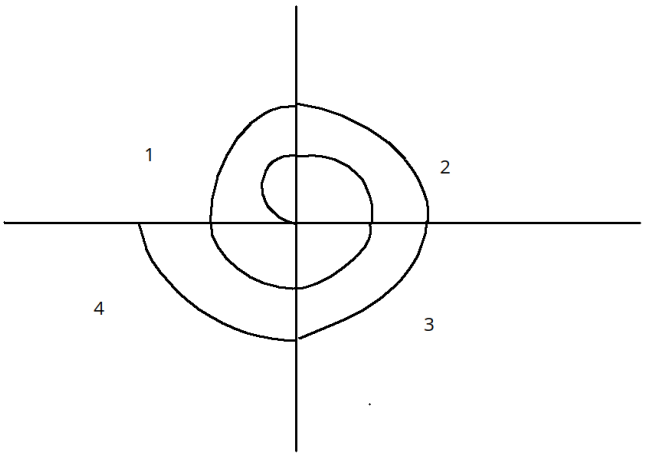
Applicability: when unknows RISKS crop up during development

One cycle (traversal through 4 quadrants) is called a PHASE. During a phase, some features of the software are added. As we move through the spiral outwards, phase by phase, the features get added.

As we move through the spiral outwards, phase by phase, new features get added.

In each phase, RISKs are evaluated, and a prototype is developed.

Thus, with each iteration around the spiral, progressively more complete versions of the of the software get built



Coding is deemphasized for a much longer period that until other models

1st Cycle:

Conceptual prototyping (Quadrant 2)

Concept of operation: System software specification (Quadrant 3)

2nd Cycle:

Demonstration Prototyping (Quadrant 2)

Software Requirement Specification (Quadrant 3)

AND

System Software specification (Quadrant 3)

3rd Cycle:

Design Assessment Prototyping (Quadrant 2)

Software Architecture (Quadrant 3)

4th cycle:

Operational Prototyping (Quadrant-2)

Simulation of Detailed Design, Code, Unit test, Integration and Test (Quadrant 3)

IOC (Initial Operational Capability) deliver (Quadrant4)

<The first build>

5th Cycle:

Updated operational prototyping (quadrant 2)
Simulation Models and benchmarks (quadrant 3)
Updated detailed design, code, unit test (quadrant 3)
Integration of Test (quadrant 3)
….
User Acceptance Test and training (quadrant 3)
F O C (Final operational capability) delivery (quadrant 4)

RISK ANALYSIS is dome in quadrant-2 of each cycle

STRENGTHS OF SPIRAL MODEL

1. Allows the users to see the system, early, though the use of rapid prototyping in the development life cycle
2. Provides early indication of insurmountable risks, without much cost.
3. Management control of quality, corrections, cost schedule and staffing is improved through reviews at the conclusion of each iteration

WEAKNESSES OF SPIRAL MODEL

1. If the project is low-risk or small, this model can be an expensive one. The time spent evaluating the risk after each spiral is costly
2. The model is complex
3. Considerable risk assessment expertise is required

WHEN TO USE THE SPIRAL MODEL

1. For the projects that represent medium to high risk
2. When the technology is new and tests of basic concepts are required.
3. When requirements are complex.
4. For large projects
5. On long projects that may make managers or customers nervous
6. When benefits are uncertain and success is not guaranteed.

Estimating DURATION AND COST

Estimating the effort, diration, and cost of software occurs diring the early stages of project planning, just after estimating the size

Effort: Amount of person-effort, or labour, that will be required to perform a task.

Units: Staff months (person-months)

COCOMO assumptions

➔ 19 productive staff days per staff month
Per staff month

➔ 152 staff hours per staff month

COCOMO

The construcive cost model -> The most widely used estimation technique

COCOMO Modes categorize the complexity of the system and the development environment.

1. **Organic mode:** The organic mode is typified by systems such as payroll, inventory, and scientific calculation. Other characterization are that the project team is small, little innovation is required, constraints and deadlines are few and the development environment is stable
2. **Semidetached mode:** The semidetached mode is typified by utility systems such as compilers, database systems, and editors. Other characterization are that the project team is medium size, some innovation is required, constraints and deadlines are moderate, and the development environment is somewhat fluid
3. **Embedded mode:** The embedded mode is typified by real time systems such as those for air traffic control, ATMs or weapon system

Cocomo levels

Three levels of detail allow the user to achieve greater accuracy with each successive level

Basic: The level uses only **SIZE** and **MODE** to determine the effort and schedule. It is useful for fast, rough estimate of small to medium size project

Intermediate: This level uses size, mode, and 15 additional variables to determine effort. The additional variables are called "COST DRIVERS" and relate to product, personal, computer and project attributes that will result n more effort or less effort required for the software project. The product of the cost derives is known as the environmental adjustment factor (EAF)

Detailed: This level builds upon intermediate COCOMO by introducing the additional capabilities of phase-sensitive effect multipliers and a three level product hierarchy. The intermediate level may be tailored to phase and product level to achieve the detailed level. An example of phase sensitive effort multipliers world be consideration of memory constraints when attempting to estimate the coding of testing phase of a project. At the same time, though, memory size may not affect the effort or cost of the analysis phase.

Phase sensitive multipliers are generall reserved for use in mature organization and require the use of an automated tool

BASIC COCOMO

Effort Estimation: KLOC is the only input variable

$Effort\ (E) = a \times (size)^b$
Where a and b are constants derived from organization analysis. (Depends on the project)
Size = thousand of lines of code (KLOC)
E = effort expressed in staff-months

The constants a and b can be determined by a curve fit procedure (regression analysis, matching product data to the equation. Most organizations do not have enough data to perform such an analysis and begin using Boehn's three levels of difficulty that seem to characterize many software projects.

Basic COCOMO Effort Formulae for three modes

Effort for Organic Mode	$E = 2.4 \times (size)^{1.05}$
Effort for Semidetached mode	$E = 3.0 \times (size)^{1.12}$
Effort for Embedded Mode	$E = 3.6 \times (size)^{1.20}$

After effort is estimated, and exponential formula is also used to calculate a project duration, or completion time (time to develop, TDEV). Project duration is expressed in months

Basic COCOMO project duration estimate

project Duration for Organic Mode	$TDEV = 2.5 \times (E)^{0.38}$
project Duration for Semidetached mode	$TDEV = 2.5 \times (E)^{0.35}$
project Duration for Embedded Mode	$TDEV = 2.5 \times (E)^{0.32}$

When effort (E) and development time (TDEV) are known, the average staff size (SS) to complete the project may be calculated.

Average Staff: $SS = Effort \div TDEV$
Productivity: $P = Size \div Effort$

RISK Management

1. Reactive Strategy: The majority of software teams rely solely on reactive strategies. At best, a reactive strategy monitors the project for likely risks. Resources are set aside to deal with them, should they become actual problems.
2. Productive strategy: A proactive strategy begins long before technical work is initiated. Potential risks are identified, their probability and impact assured, and they are ranked by importance. Then the software team establishes a plan for managing risks. The primary objective is to avoid risk, but because all risks cannot be avoided, the team works to develop a contingency plan that will enable to respond in a controlled and effective manner.

Software Risks

Risk involves two characteristics:

i. Uncertainity: risk may or may not happen
ii. Loss: if the risk becomes a reality, unwanted consequences or losses will occur.

Risk exposure can be computed for each risk in the risk table

$$RE = P \times C$$

Where RE = Risk exposure, P = probability of occurance of the risk, C = cost to the project if the risk occur

Example:

Risk Identification: Only 70% of the software components scheduled for reuse will in fact be integrated into the application. The remaining functionality will have to be custom developed.

Risk Probability: 80%

Risk Impact

No of reusable software component = 60

….. components to be deploed from scratch = (100% - 70%) x 60 = 18

Average size of component = 100 LOC

Software engineering cost for each LOC = $14.00

Overall cost to develop the components: $C = 18 \times 100 \times \$14 = \$25.000$

Risk Exposure RE = P X C = (0.80) x $25,200 = $20,000

Risk Management plan

Step 1: Constuct risk categorzaton table (starting point for identification of specific risk

RISK FACTORS and CATEGORIES	L-low risk evidence	M-Medium Risk Evidence	H-High Resk Evidence	Rating (H/M/L)	Comments
1. MISSION and GOAL factors 2. Organization Management Factors 3. Customer Factors 4. Budget/Cost factors 5. Schedule Factory 6. Project Content 7. Performance Factory 8. Project Management Factors 9. Development Process Factory 10. Development Environment Factor 11. Staff Factor 12. Maintenance Factors.			<Characterisics of the factor when risk is high>	<Select the level of risk applicable>	<Provide information about project specific that justifies the rating as H, M, or L

Step 2: Rank the risk to the project for each category (Fill-in the last two columns)

Step 3: Sort the risk table in order of risk withhigh-risk items first . For the top ten risks, and all risks rated "high" (if more that ten), calculate the risk exposure. These are the key risks.
For each key risk,

- ➔ Identify means to control the risk
- ➔ Establish ownership of the action
- ➔ Establish date of completion.

Integrate the key risks into the project plan and determine the impacts on schedule and cost.

Step 4: Show the report**,** establish a regular risk report format for weekly project status meeting

Risk Item	Rank this week	Host rank	Number of weeks on list	Resolution Approch
Too few engineering experts	1	1	2	Contract under discussion
Design Schedule tight	2	2	2	Enforcing Delphi estimates
…				
New technology	10	10	4	Reviewed requirement

RISK RESPONSE TABLE (or "RISK REGISTER")

ID (Sl. No)	Risk Item	Trigger (threshold which of exceeded required action)	Value	Risk Exposure	Resolution Approach	Who is responsible for the action	Date (Action due)

Risk Management Plan (contd.)

Step 5: The final step is to ensure that risk management is an ongoing process within the project management. Monitoring and control of the risk list must be done on a regular basis. The project manager and team must be aware of the identified risk and the process for resolving them. New risks must be identified as soon as possible, prioritized and added on to the risk management plan. High priority risks must be worked on with respect to the overall project plan.

SOFTWARE QUALITY ASSURANCE

Quality assurance (QA) is a preventive approach.
Quality Control (QC) is a connective approach.

ISO 9000 Definition
Quality control is the operational techniques and activities that are used to fulfil the requirements for quality. QC -> refers to the quality related activities with the creation of project deliverables.

Quality assurance means all those planned and systematic activities implemented to provide adequate confidence that an entity will fulfil requirements for quality.
QA -> refers to the process used to create the deliverables. (It is generic; it does not concern the specific requirements of the product being developed)

Metrics for Object oriented Software

"C K Metrics" -> Chidamber and Kermerer, IEEE "Trans in software engineering Vol 20, No. 6, pp 476-493, June 1994

CK-1: Weighted Methods per Class (WMC)

Complexity ≡ Cyclomatic Complexity (of a method)
For class A, with methods M_1, M_2, … M_n with complexitites C_1, C_2, … C_n respectively,

$$WMC = \sum_{i=1}^{n} C_i$$

Since measuring Cyclomatic Complexity is difficult, WMC is ofter measured simply by the number off methods in a class. A desirable value should be less than 20.

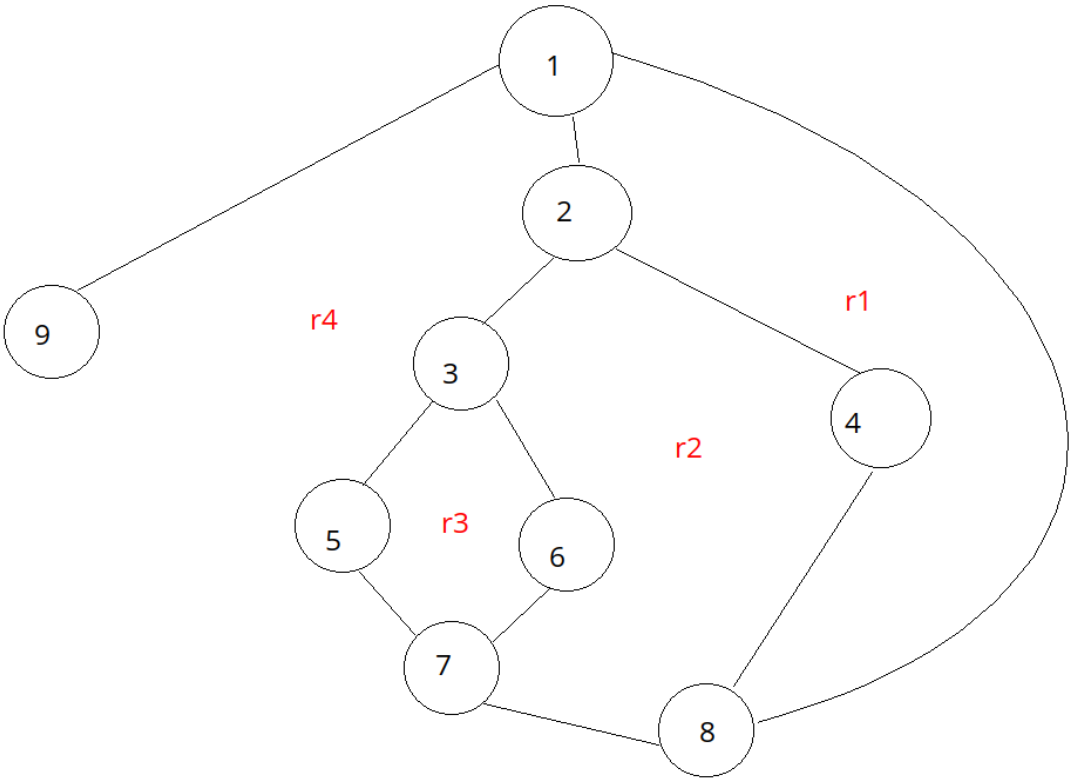
Cyclomatic Complexity V(G) of a graph G: There are three ways to compute V(G)

1. V(G) = E – N + 2

   Where N= number of nodes, E = number of edges
2. V(G) = R,

   Where R is the number of regions. (Area bounded by edges and nodes is called a region. Area outside the graph is also a region)
3. V(G) = P + 1

   Where P = number of binary decision nodes contained in the control flow graph.



CK-2: Depth of Inheritance Tree (DIT)

DIT = Maximum length from node to root of tree. DIT should be less than 6

CK-3: Number of Children (NOC) {of a class}

NOC = number of immediate sub-classes. It should have a low value

CK-4: Coupling between Object classes (CBO)

Two classes are coupled when methods declared in one class use methods or instance variables of the other class.

For a given class,

CBO ≡ number of other classes with which this class is coupled. It should be a low value

CK-5: Response for a class (RFC)

RFC ≡ number of methods that can be executed in response to a message received by an object of that class. Large RFC has been found to indicate more faults.

CK-6: Lack of Cohesion in Methods (LCOM)

LCOM ≡ number of methods in a class that reference a given instance variable.

High cohesion indicates good class subdivision.
Low cohesion increases complexity
LCOM has been criticized. Several alternatives have been developed.

Coupling

There is empirical evidence to support the benefits of LOW coupling between objects/classes/software modules.

The STRONGER the coupling between software modules,

i. The more difficult it is to understand individual modules and hence to maintain or enhance them correctly
ii. The larger the sensitivity of (unexpected) change and defect propagation effects across modules; and
iii. The move is the testing required to achieve satisfactory reliability levels.

$$Coupling\ Factor\ (CF) = \frac{Number\ of\ non - inheritance\ couplings}{maximum\ number\ of\ couplings}$$

Cohesion:

Cohesion refers to how closely the operations in a class are related to one another.
Cohesion of a class is the degree to which the local methods are related to the local instance variables in the class.

Methods of measuring cohesion

1. Calculate for each data field of a class what percentage of the methods use that data field. Average the percentage, then subtract from 100%. Lower percentage means great cohesion of data and methods in the class.
2. Methods are more similar if they operate on the same attributes. Count the number of disjoint sets produces from the interaction of the sets of attributes used by the methods.