

SE FINAL NOTES

BEFORE MID 2

CHAP 1-3(somerville)

SE and its concepts:

Software Engineering (SE) is a discipline that focuses on the systematic development, operation, and maintenance of software systems. It encompasses a range of principles, methods, tools, and practices aimed at creating high-quality software that meets user requirements and is reliable, efficient, and scalable. Here are some key concepts within software engineering:

1. Software Development Life Cycle (SDLC): A process framework for developing, operating, and maintaining software systems.
2. Requirements Engineering: Eliciting, analyzing, and managing user needs and system requirements.
3. Software Design: Creating the architectural and detailed design of the software system.
4. Software Testing: Evaluating software quality and correctness through testing and debugging.
5. Software Maintenance: Managing and enhancing software systems throughout their lifecycle.
6. Software Quality Assurance (SQA): Ensuring software products and processes meet established quality criteria.

7. Software Configuration Management: Managing changes to software artifacts, version control, and release management.
8. Software Project Management: Planning, organizing, and controlling resources to achieve project goals.
9. Software Documentation: Creating and maintaining documentation for understanding and communicating software systems.
10. Software Engineering Ethics: Adhering to professional and ethical standards in software development and use.

Case studies:

A scenario is given and it is asked to identify components such as functional, non functional requirements, flow charts, etc.

Plan driven vs agile:

Plan-Driven Methodology:

- Sequential and linear approach
- Emphasis on thorough planning and documentation
- Fixed scope, timeline, and budget
- Limited customer involvement and flexibility
- Documentation-heavy

Agile Methodology:

- Iterative and incremental approach
- Flexibility and adaptability to changing requirements
- Collaborative and customer-centric
- Continuous improvement and reflection
- Emphasis on working software over extensive documentation

Software process models:

Waterfall Model: Linear and sequential approach with distinct phases (requirements, design, implementation, testing, deployment, maintenance).

Incremental Model: Software development divided into smaller iterations, delivering working software incrementally.

Configuration and Integration: Managing and combining software components to create a functioning system, focusing on configuration control and integration testing.

Reuse built in projects:

Reusing built-in projects means leveraging existing software components to save time and improve reliability. It involves using pre-developed solutions for common functionalities, enhancing productivity and customization.

IMP model and its design activities:

IMP (Information Modeling and Processing) Model:

An approach for information systems development focusing on information modeling and processing.

Design Activities:

Creating a blueprint for the software system, including:

- architectural design
- detailed design
- user interface design
- database design
- security design
- integration design
- testing design

Software validation:

Software validation ensures that the software meets user requirements and functions correctly. It involves activities like testing, compliance with standards, and evaluating the software from the user's perspective. It is an

iterative process performed throughout development, supported by documentation.

- Objectives
- Activities
- Compliance
- User perspective
- Validation vs verification
- Iterative process
- Documentation

Alpha and beta testing:

Alpha Testing:

- Alpha testing is conducted in the early stages of software development, typically within the development environment.
- It is performed by the internal team or a select group of users closely associated with the development process.
- The main focus is on identifying bugs, issues, and functionality problems before releasing the software to a larger audience.
- Alpha testing helps gather feedback, make necessary improvements, and ensure stability and reliability of the software.

Beta Testing:

- Beta testing takes place after alpha testing and involves a wider audience of external users who are not directly involved in the development process.
- The software is released as a beta version to gather real-world usage scenarios, feedback, and identify any remaining bugs or issues.
- Beta testers provide valuable insights into usability, performance, compatibility, and other factors that help refine the software.
- The aim is to gather diverse feedback, validate the software in different environments, and make final adjustments before the official release.

Alpha testing is performed internally by the development team or a select group to uncover issues early on, while beta testing involves external users

to obtain feedback and validate the software in real-world scenarios before the official release.

Testing stages:

Testing Stages:

1. **Unit Testing:** Testing individual components or modules of the software to ensure they function correctly in isolation. It focuses on verifying the smallest units of code.
2. **Integration Testing:** Testing the interaction and integration between different components or modules to ensure they work together seamlessly. It aims to identify any interface issues, data flow problems, or compatibility concerns.
3. **System Testing:** Testing the entire system as a whole to verify that it meets the specified requirements. It focuses on evaluating the system's behavior, functionality, and performance as a complete entity.
4. **Acceptance Testing:** Testing the software from the end-user's perspective to determine if it meets their expectations and requirements. It may include user acceptance testing (UAT) conducted by the client or stakeholders.
5. **Regression Testing:** Repeating previous tests to ensure that changes or fixes in the software have not introduced new defects or caused unintended consequences. It helps maintain the stability and integrity of the software after modifications.
6. **Performance Testing:** Evaluating the software's performance and responsiveness under expected or peak workload conditions. It involves measuring factors such as speed, scalability, reliability, and resource usage.

7. Security Testing: Assessing the software's security measures to identify vulnerabilities, weaknesses, and potential risks. It involves penetration testing, vulnerability scanning, and validating the effectiveness of security controls.

8. User Interface (UI) Testing: Testing the user interface components to ensure usability, accessibility, and visual consistency. It focuses on user interactions, navigation, responsiveness, and adherence to design guidelines.

Software prototyping:

Software Prototyping: Iterative development approach involving creating a simplified version of the software to gather feedback, refine requirements, and improve design before final development.

Approaches to improvement:

Approaches to software improvement focus on enhancing the quality, efficiency, and effectiveness of software development and maintenance processes. Here are some concise approaches:

1. Continuous Improvement
2. Process Automation
3. Agile Methodologies
4. Quality Assurance
5. Measurement and Metrics
6. Adoption of Best Practices
7. Feedback and Collaboration

IMP read capability maturity levels:

1. Level 1 - Initial: Processes are ad hoc and unpredictable, lacking standardization. Success depends on individual effort, and there is limited process awareness or documentation.
2. Level 2 - Managed: Basic project management processes are established, aiming to bring stability and control. Processes are documented, repeatable, and tailored to specific projects.
3. Level 3 - Defined: Standardized and documented processes are established across the organization. Processes are well-understood and consistently applied. There is a focus on process optimization and institutionalizing best practices.
4. Level 4 - Quantitatively Managed: Quantitative objectives and measurements are defined to control and manage process performance. Statistical techniques are used to analyze and make data-driven decisions for process improvement.
5. Level 5 - Optimizing: Continuous process improvement is ingrained in the organization's culture. Processes are monitored, measured, and optimized proactively. Innovation and new approaches are actively sought to drive higher performance.

Each maturity level builds upon the previous one, representing an evolutionary progression towards higher levels of process maturity, consistency, and optimization.

Extreme Programming:

Extreme Programming (XP) is an agile methodology for software development that emphasizes: customer collaboration
iterative development

test-driven development
continuous integration
teamwork
frequent feedback
customer-centric

Refactoring:

Refactoring is the practice of improving code without changing its external behavior. It involves making changes to enhance code quality, readability, and maintainability. By applying refactoring techniques, developers improve code structure and efficiency without altering its functionality.

Test first development:

Test-First Development, also known as Test-Driven Development (TDD), is an approach to software development that emphasizes writing automated tests before writing the actual code to ensure that the code meets the desired functionality and passes the defined tests.

Software method and maintenance:

Software Method: An approach or methodology used in software development to guide the overall process, including requirements gathering, design, implementation, testing, and maintenance. Examples of software methods include Waterfall, Agile, Scrum etc.

Software Maintenance refers to the process of modifying, enhancing, and updating software after its initial development and deployment. It involves making changes to the software to correct defects, improve performance, adapt to changing requirements, and address customer feedback.

Agile maintenance:

Agile Maintenance: Applying agile principles and methodologies to manage and deliver software maintenance activities. It involves embracing iterative and incremental approaches, close collaboration with stakeholders, continuous improvement, and adaptability to changing requirements during the maintenance phase. Agile maintenance promotes flexibility, responsiveness, and efficient delivery of enhancements, bug fixes, and updates to existing software systems.

Factors in large systems:

Factors in large systems that can impact design, development, and management:

1. Scalability: Handling increasing volumes efficiently.
2. Modularity: Breaking down into manageable components.
3. Performance: Meeting response time requirements.
4. Security: Protecting sensitive data.
5. Integration: Seamless connection with other systems.
6. Maintenance and Upgrades: Easy maintenance and smooth upgrades.
7. Collaboration and Communication: Effective teamwork and coordination.
8. Documentation: Comprehensive reference materials.

Scrum and cycle:

Scrum and the Software Development Life Cycle (SDLC) are two different approaches used in software development projects. Here's a concise comparison:

Scrum: Scrum is an agile framework that focuses on iterative and incremental development. It divides the project into short iterations called "sprints," typically lasting 1-4 weeks. Scrum promotes frequent collaboration, self-organizing teams, and adaptive planning. It emphasizes flexibility, continuous improvement, and delivering value to customers throughout the development process.

Software Development Life Cycle (SDLC): The SDLC is a structured approach that encompasses the entire software development process, from initial planning to deployment and maintenance. It typically follows a linear or sequential approach, where the project progresses through distinct phases such as requirements gathering, design, development, testing, and deployment. SDLC frameworks like Waterfall and V-Model are examples of linear approaches.

In summary, Scrum is an agile framework that operates within the broader context of the SDLC, which encompasses various methodologies and models used in software development projects.

Multi team scrums:

Multi-team Scrum: Multi-team Scrum, also known as Large-Scale Scrum (LeSS) or Scaled Agile Framework (SAFe), is an approach used when multiple teams collaborate on a single project or product. Coordination, synchronization, and collaboration across teams to ensure project success.

In multi-team Scrum, the principles and practices of Scrum are adapted to accommodate the larger scale and complexities of the project. The goal is to maintain the agility and benefits of Scrum while enabling multiple teams to work in parallel and deliver a cohesive product.

Agile methods across organizations:

Agile Methods across Organizations: Implementing agile principles and practices at an organizational level to drive agility, flexibility, and responsiveness. It involves aligning processes, structures, and culture to support iterative development, continuous improvement, and customer-centricity. Agile methods across organizations promote collaboration, adaptability, and faster delivery of value to customers.

AFTER MID 2

Quality management chap 24(somerville)

Software quality:

- User needs are fulfilled (functional quality)
- Reliable and efficient performance (structural quality)
- On time and within budget delivery (process quality)
- Measured in terms of customer satisfaction

SQ management:

Quality:

- Assurance: Ensuring a high-quality product through standardized processes, verification, and validation by the development team.
- Planning
- Control: testing the products and identifying defects to improve them.
- Improvement: improving the product quality with the changing requirements

Quality management(QM) is important at organizational level(framework for high quality software) and project level(for quality project).

QM team checks:

- the project deliverables to ensure consistency
- The documentation to verify completion of tasks
- assumptions made by one team without communicating it to others
- Product's release testing(in large companies ONLY)

The QM team should be independent from the development team.



Quality planning:

Quality Plan: Outlines desired product qualities, assessment methods, and defines key quality attributes. It should be short and to the point.

Outline structure for quality plan includes:

- product introduction
- Product plans
- Process descriptions
- Quality goals
- risks identification & risks management

QM is vital for large and complex systems.

Quality documentation ensures continuity despite team changes.

Software quality is dependent on functional and nonfunctional requirements.

Software quality attributes:

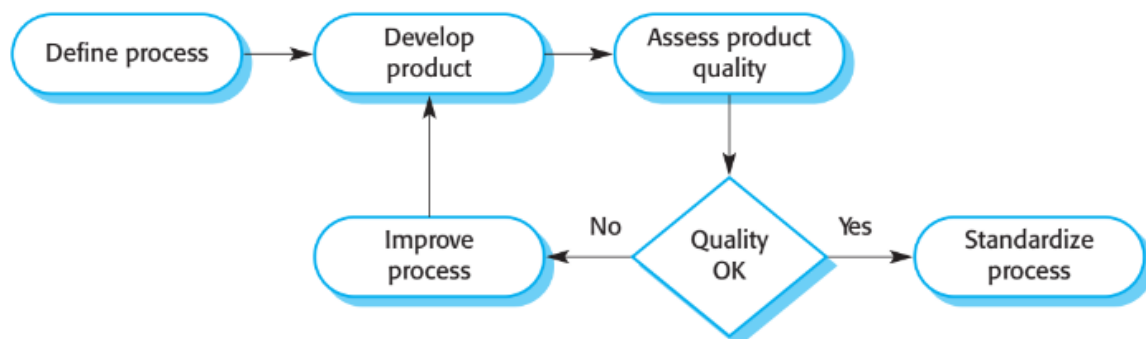
Safety	Understandability	Portability
Security	Testability	Usability
Reliability	Adaptability	Reusability
Resilience	Modularity	Efficiency
Robustness	Complexity	Learnability

It is not possible for a system to be optimized so quality conflict can occur.

Software process and product quality:

The quality of a product \propto the quality of the production process.

Process based quality assessment:



Quality culture:

- Everyone responsible for software development, is committed to achieving a high level of product quality
- Encourage teams to take responsibility for the quality of their work
- Support people who are interested in improvement of intangible(non materialistic) aspects

Software standards:

- Can be external(national, international) and internal
- Provide continuity of work, a framework, and encapsulate best practices
- Types :
 - Product
 - Document standards
 - Documentation standards

- Coding standards
- Applied to software product being developed
- Process
 - Followed during software development
 - Develop a better end product
 - Focus on design and validation
 - Description of documents

Problems with standards

- Outdated
- Too much formality
- inadaptability

Quality management and agile development:

Agile development is opposed to QM due to documentation overheads

Agile development uses some shared practices instead of documentation:

- Check before check in(check your own work before turning in)
- Never break the build(don't integrate failed code)
- Fix problems when you see them

For large systems built for external customer agile development seems impractical.

[Chapter 8: Software testing](#)

- Testing is the process of showing what a program is doing.
- It is also used to detect anomalies in the system.
- In custom software each requirement has atleast 1 test and in generic software each feature present has at least 1 test case.
- Testing is concerned with rooting out undesirable system behavior such as system crashes, unwanted interactions with other systems, incorrect computations, and data corruption.
- Testing can be manual or automatic.
- Part of Software verification and validation.
- There are 2 types of testing:

- Black box testing: testing system without checking inner details
- White Box testing: testing each component separately.
- Black-box testing is typically faster than white-box testing.
- Black-box testing is typically less expensive than white-box testing.
- Black-box testing can be performed by testers with a lower skill level than white-box testing.
- White-box testing goes into more depth than black-box testing, as it can reveal sensitive information about the software.
- A software can be tested for both functional(code and functionality) and non-functional(scalability, security and performance) requirements.

Validation and verification, what is the fu*king difference?

	Verification	Validation
Definition	It is a process of checking if a product is developed as per the specifications.	It is a process of ensuring that the product meets the needs and expectations of stakeholders.
What it tests or checks for	It tests the requirements, architecture, design, and code of the software product.	It tests the usability, functionalities, and reliability of the end product.
Coding requirement	It does not require executing the code.	It emphasizes executing the code to test the usability and functionality of the end product.
Activities include	A few activities involved in verification testing are requirements verification, design verification, and code verification.	The commonly-used validation activities in software testing are usability testing, performance testing, system testing, security testing, and functionality testing.
Types of testing methods	A few verification methods are inspection, code review, desk-checking, and walkthroughs.	A few widely-used validation methods are black box testing, white box testing, integration testing, and acceptance testing.
Teams or persons involved	The quality assurance (QA) team would be engaged in the verification process.	The software testing team along with the QA team would be engaged in the validation process.

Target of test	It targets internal aspects such as requirements, design, software architecture, database, and code.	It targets the end product that is ready to be deployed.
-----------------------	--	--

What is the difference between validation and verification?

Types of Bugs/Errors	Software Inspection	Software Testing
Checks the complexity of Code	Yes	No
Reviews Code Structure	Yes	No
Identifies interface defects at module level	Yes	No
Checks if new features are to be added	Yes	No
Identifies and resolves boundary level defects	Yes	Yes
Resolve performance errors	Yes	Yes
Checks if any features are implemented incorrectly	Yes	Yes

Stages of testing:

Development testing:

- Done by developers during development
- There are 3 types:
 - Unit testing:
 - Focuses on testing individual components in isolation.
 - Units are defined by developers

- They can be functions, objects, classes, a combination of everything or a separate module.
- It is more efficient if it is automated, with the help of frameworks.
- Important to define test cases to run properly, meaning it should correctly function when given the correct input and identify defects.
- Interface testing:
 - It is the testing done at the interaction point of 2 units.
 - Cannot be identified by unit testing
- Component testing:
- System testing:
 - Testing of the integration of multiple components
 - Tests interaction between 2 or more components
 - Separately tested components are integrated with newly developed components to be tested together.
 - Done by development team

Release testing:

- The process of testing a particular release or version of a system that is intended to be used outside of the development team.
- Usually black box testing.
- Done by a separate team that was not involved in the system development.
- The objective of release testing is to check that the system meets its requirements and is good enough for external use (validation testing).

User testing:

- Customers test the software
- Process of formally testing a system provided by the supplier.
- Users experience the new software product and identify whether the system is doing the intended work, check for interface and defects.
- Benefits:
 1. Identifies usability problems
 2. Gather's user's feedback
 3. Helps in understanding user needs
 4. Improves the user experience

Chapter 8: Design concepts:
Chapter 69(11): UI Design

3 golden principles:

1. Allow users to maintain control:
 - a. Give the user the illusion of control
 - b. Allow flexible interaction
 - c. Hide technical working from user
 - d. Design for direct interaction with on screen items
2. Reduce user's memory load
 - a. Reduce learning curve
 - b. Establish functions which multiple pre built uses
 - c. Have intuitive shortcuts
 - d. Disclose information in a progressive fashion
3. Make consistent interface design decisions:
 - a. Maintain consistent design features such as colour schemes and patterns
 - b. Do not make changes to the software such that they conflict with the user's habits and preferences.

UI Design Process:

1. **User Research:** Research user needs.
2. **Sketching and wireframing:** creating rough sketches
3. **Visual Design:** Developing a visual design of the interface
4. **Prototyping:** create a working prototype
5. **Iteration:** Continuous refinement, based on user feedback.

Use this acronym to remember it: **U**nicorns **s**wiftly **v**aulted **p**ast **i**llusions

UI Analysis:

1. Usability testing: testing the effectiveness of the interface.
2. Task analysis: Analysing user tasks and behaviours to identify areas for improvement in the interface.
3. Heuristic Evaluation: testing the interface against standards
4. User feed: taking user feedback

UI Evaluation:

1. Performance Metrics: Measuring performance metrics such as loading time, error rates, and conversion rates to evaluate the effectiveness of the interface.

2. A/B Testing: Testing two different versions of the interface to determine which is more effective.
 3. Analytics: Analyzing user data such as click-through rates, engagement, and bounce rates to identify areas for improvement.
 4. Comparative Analysis: Comparing the interface to competitors' interfaces to identify strengths and weaknesses.
-

Estimation for software projects chap26 (Pressman)

Project estimation is done on the basis of:

- Project size (LOC)
- Project cost (\$)
- Effort required (person month)
- Project duration (months/days)

Metrics used for project size estimation:

- LOC (Line Of Code)
- FP (Functional Point)

LOC based estimation:

- Total LOC
- Effort required=total LOC/LOC per person month
- Project cost=total LOC*cost per person month(effort required)
- Project schedule(duration)=effort required/number of developers

- Login and registration module: 1000 LOC
- Search and booking module: 3000 LOC
- Payment and checkout module: 2000 LOC
- User profile module: 1500 LOC
- Feedback and rating module: 500 LOC
- Use historical data to estimate the effort required to develop the project: Let's assume that the historical data suggests an average of **600 LOC per person-month** and an average cost of **\$8000 per person-month**.
- Calculate the total number of lines of code:
- Total LOC = 1000 + 3000 + 2000 + 1500 + 500 = **8000**
- Effort required = Total LOC / LOC per person-month = 8000 / 600 = **13.3 person-months**.
- Project cost = Estimated LOC* Cost per person-month = **\$8000 * 13.3 = \$106,400**
- Project schedule = Effort required / Number of developers = 13.3 / 2 = **6.65 months**

FP based estimation:

- UFP(Unadjusted Function Points)= calculated by multiplying each function type with its corresponding weighting factor and add them
- CAF(Complexity Adjustment Factor)=14 questions are answered(on a scale of 0-5) based on complexity and added($\sum F_i$).
- $CAF = 0.65 + (0.01 * F_i)$
- $FP = UFP * CAF$
- 14 questions:
 - No influence/not important=0
 - Incidental=1
 - Moderate=2
 - Average=3
 - Significantly complex=4
 - essential=5

- User input = 55
- User outputs = 35
- User enquiries = 40
- User files = 8
- External interfaces = 5
- Calculate FP = UFP * CF

Measurement parameter	Counts	Low	Average	High	=count * weighting factor
External inputs	55	3	4	6	?
External outputs	35	4	5	7	?
External enquired	40	3	4	6	?
Internal logic files	8	7	10	15	?
External interface files	5	5	7	10	?
Count total					= ?

Calculate for high complexity

- Calculate CAF

$$CAF = 0.65 + (0.01 * \Sigma Fi)$$

- Lets say, the product is significantly complex product so all 14 questions answered in moderately form i.e. 4

$\Sigma Fi = ?$ (if complexity factor is differing for each FP then calculate separately.)

$$CAF = 0.65 + (0.01 * ?) = ?$$

$$FP = UFP * CAF$$

$$FP = ? * ? = ?$$

Complexity factors

5 questions = Average

5 questions = Moderate

4 questions = No influence

$$\Sigma Fi = (5 \times 3) + (5 \times 2) + (4 \times 0) = 25$$

Chapter 27 estimation: activity diagrams and stuff

Project planning:

- It involves Scope definition(WBS): which just means identifying the project's
 - Goals: what are the aims of the project
 - Deliverables: products and services produced
 - Task: how to develop deliverables
 - Boundaries: limits
- Software estimation
- Software scheduling
- Involves splitting project into tasks
- Estimation of the cost, time and resources required
- Organization of tasks concurrently to make optimal use of resources and workforce
- Minimizing dependencies between tasks
- Dependent on project managers intuition and experience.
- Scheduling problems:
 1. More number of people does not mean an increase in productivity
 2. Including more people in a project that is already running behind schedule causes further delays due to the increased need for communication and coordination.
 3. Always have a contingency plan



Techniques:

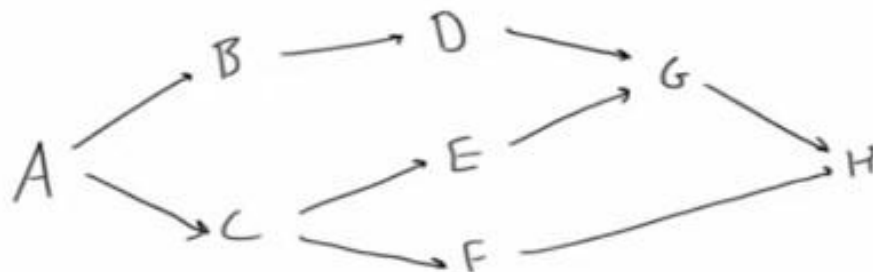
- Bar charts can be used, by plotting task time against the calendar time.

Activity Diagram:

- To make one first u need a table:


Activity	Predecessor	Duration (days)
A	-	3
B	A	4
C	A	2
D	B	5
E	C	1
F	C	2
G	D,E	4
H	F,G	3

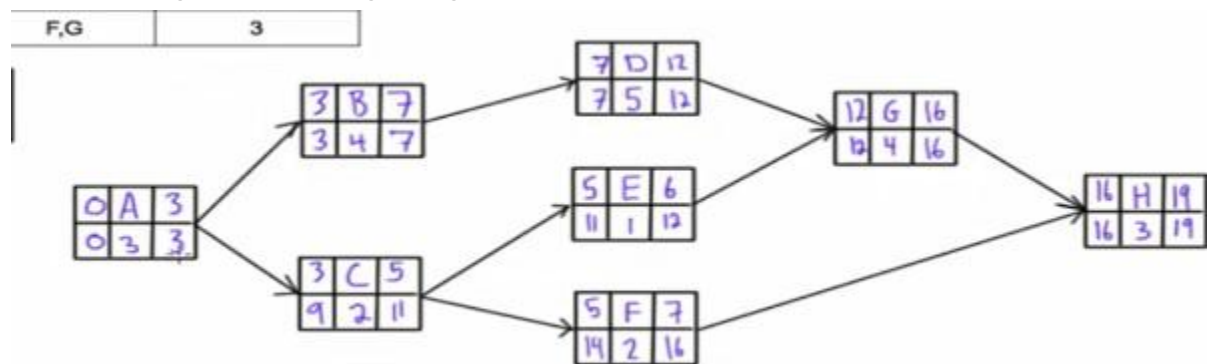
- Now we need to construct a simple graph showing all the dependencies between each activity. Like this:



- Now we need to add a special node with 6 boxes, like this:

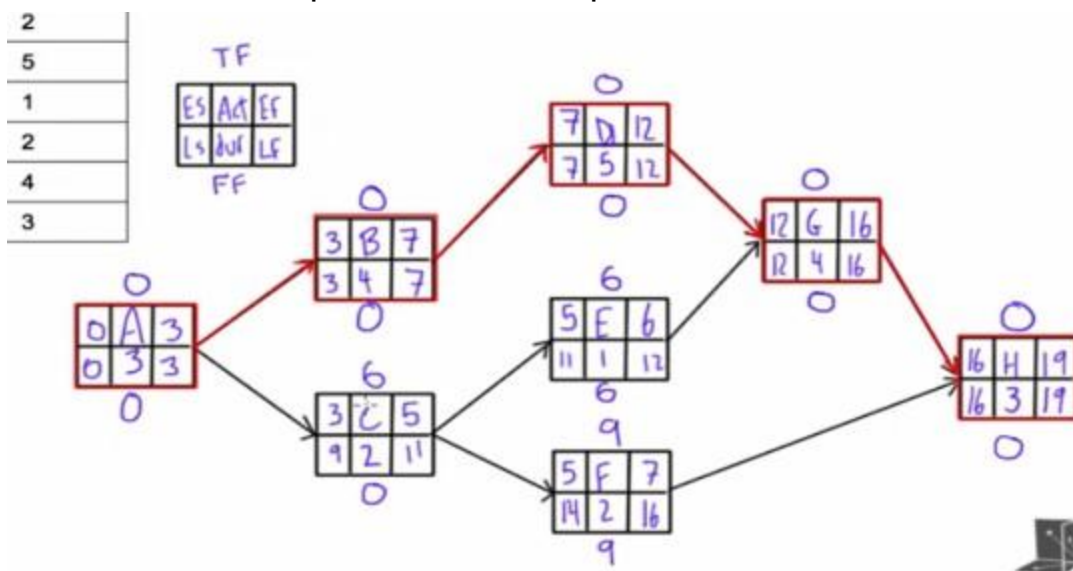
Early start(ES)	Write the name of activity here	Early Finish(EF)
Late start (LS)	Write the duration here	Late Finish(LF)

- Replace each vertex with these nodes ,
- Calculating ES and EF(forward passing):
 - By default the ES of the starting node is 0
 - $EF = ES + \text{duration of the node}$
 - If an activity branches out into multiple activities, then all of those activities will have the same ES.
 - If 2 or more activities merge into one activity, then the ES of that activity will be greatest of the previous nodes.
 - ES of all nodes ,except the first node, are the EF of the previous node
 - Calculating these we get (ignore the last row)



-
- Calculating LS and LF(backward passing):
 - Unlike the calculation of ES and EF, LS and LF are calculated from the end node.
 - By default the LS and LF of the final node are the same.
 - If an activity branches out into multiple activities, then all of those activities will have the same LF.
 - If 2 or more activities merge into one activity, then the LF of that activity will be smallest of the previous nodes.
 - To calculate the $LS = LF - \text{duration}$.

- Total float: is how long an activity can be delayed without putting off the project completion date.
 - Calculate it using $\text{Total Float} = \text{Late Start} - \text{Early Start}$ OR $\text{Total Float} = \text{Late Finish date} - \text{Early Finish date}$
- Free float: is how long an activity can be delayed without delaying the Early Start of its successor.
 - Calculate it using: $\text{Free Float} = \text{ES of next Activity} - \text{EF of Current Activity}$
 - Free float of the last node is 0
- If Total float(TF) and Free float(FF) are both 0, that is a critical path.
- An activity diagram might have multiple or none.
- Here is the critical path of our example:



Ricks management chap 28(Pressman)

Actions taken to understand and manage uncertainty. It is performed by everyone involved in the software process.

Reactive ricks strategy: responding to ricks after they occur. It can lead to crisis management.



To avoid this we use

Proactive ricks strategy: Potential ricks are identified, their probability and impact are assessed, and they are ranked by importance. Then, the software team establishes a plan for managing ricks.

Software ricks:

- Uncertainty= ricks may or may not happen
- loss= unwanted consequences

Types of software ricks:

- Project risks
 - Threatens project plan
 - Budget, personnel, schedule, resources, stakeholders, requirements, project complexity, project size
- Technical risks
 - Threatens quality and timeline of project
 - Design, interface, implementation, verification, maintenance, technical uncertainty, specification ambiguity
- Business risks
 - Threatens viability of the project(ability to be successful)
 - Building a product so good that no one really wants(market risks)
 - Product that no longer fits(strategic risks)
 - Product that the sales team can't sell(sales risks)
 - Losing support of senior management(management risks)
 - Losing budget/personnel(budget risks)

In summary, project risk relates to the specific risks associated with project execution and deliverables, technical risk focuses on uncertainties and challenges related to the technical aspects of a project, and business risk encompasses broader risks that impact the overall success and sustainability of an organization.

There can also be:

- Known risks
- Predictable risks
- Unpredictable risks

Risk identification:

For each category there are two general types of risks:

- **Generic risks** are a potential threat to every software project
- **Product-specific risks** can be identified only by those with a clear understanding of the technology, the people, and the environment that is specific to the software that is to be built

A checklist can be created to identify risks:

- Product size
- Business impact

- Stakeholder characteristics
- Process definition
- Development environment
- Technology to be built
- Staff size and experience

Questions can be created to assess overall risk.

Software risks components:

- Performance
- Cost
- Support
- Schedule

These components are driven by risk drivers, and their impact can be:

- Negligible
- Marginal
- Critical
- Catastrophic

Risk projection/estimation:

- the likelihood or probability that the risk is real
- the consequences of the problems associated with the risk if it occurs

For risk projection:

- Establish a scale
- Outline the consequences
- Estimate the impact
- Assess the overall accuracy of the risk projection

Develop a risk table:

Risks	Category	Probability	Impact	RMMM
Size estimate may be significantly low	PS	60%	2	
Larger number of users than planned	PS	30%	3	
Less reuse than planned	PS	70%	2	
End-users resist system	BU	40%	3	
Delivery deadline will be tightened	BU	50%	2	
Funding will be lost	CU	40%	1	
Customer will change requirements	PS	80%	2	
Technology will not meet expectations	TE	30%	1	
Lack of training on tools	DE	80%	3	
Staff inexperienced	ST	30%	2	
Staff turnover will be high	ST	60%	2	
Σ				
Σ				
Σ				

The consequences of risk is assessed by:

- Nature
- Scope
- Timing

Risk mitigation, monitoring and management(RMMM):

- Using proactive approach develop a risk mitigation plan
- Monitor the project risks
- Monitor effectiveness of risk mitigation plan
- If risk occurs use risk management
- Refocus resources to to those functions that are under risk

RMMM is properly documented.

It can increase budget and duration of the project
