PROJECT QUALITY MANAGEMENT
PROJECT SCHEDULING

Dr. SELİN METİN
Orion Innovation Türkiye

Overview

- Quality Management
  - Quality Planning
  - Quality Assurance
  - Quality Control
- Project Scheduling
  - Work Items
  - Results and Timing
  - Dependencies
  - Resources
  - Duration
  - Constructing the Project Network Diagram
  - Dependency Relationships and Constraints
  - Paths Through the Network
  - Compressing the Schedule
  - Management Reserve

Quality Management – Definition of Quality

- Fit for use
- Meets all client requirements
- Delivered on time within budget and according to client specifications

Types of Quality

- Process Quality
  - The quality of the project management process that produced the product
- Product Quality
  - The quality of the deliverables from the project

Quality Management

- Quality management complements project management. Important aspects are
  - Customer satisfaction – understanding, managing, and influencing needs so that customer expectations are met.
  - Prevention over inspection – cost of prevention is much lower than cost of correction
  - Management responsibility – success requires participation of all members of the team but management is responsible to provide resources
  - Processes within phases – **repeated plan-do-check-act cycle**

What is Software Quality?

- **Software Quality** is the degree of conformance to explicit or implicit requirements and expectations.
  - Explicit: clearly defined and documented
  - Implicit: not clearly defined and documented but indirectly suggested
  - Requirements: business/product/software requirements
  - Expectations: mainly end-user expectations

- Definition by IEEE
  - The degree to which a system, component, or process meets specified requirements.
  - The degree to which a system, component, or process meets customer or user needs or expectations.

- Definition by ISTQB
  - Quality: The degree to which a component, system or process meets specified requirements and/or user/customer needs and expectations.
  - Software Quality: The totality of functionality and features of a software product that bear on its ability to satisfy stated or implied needs.

The Worst Computer Bug in History: Therac-25

- The Therac-25 was a computer-controlled radiation therapy machine produced by Atomic Energy of Canada Limited (AECL).

- It was involved in at least six accidents between 1985 and 1987, in which patients were given massive overdoses of radiation. At least two patients died of the direct consequences of the overdoses.

- These accidents highlighted the dangers of software control of safety-critical systems, and they have become a standard case study in health informatics and software engineering.

Therac-25: Problem Description

- Two software faults were to blame:
  - When the operator incorrectly selected X-ray mode before quickly changing to electron mode, which allowed the electron beam to be set for X-ray mode without the X-ray target being in place.
  - A second fault allowed the electron beam to activate during field-light mode, during which no beam scanner was active or target was in place.

- Previous models had hardware interlocks to prevent such faults, but the Therac-25 had removed them, depending instead on software checks for safety.
  - Using software instead would in theory reduce complexity, and reduce manufacturing costs.

Therac-25: Root Causes

- A commission concluded that the primary reason should be attributed to the bad software design and development practices, and not explicitly to several coding errors that were found. In particular, the software was designed so that it was realistically impossible to test it in a clean automated way.
  - AECL did not have the software code independently reviewed.
  - AECL did not consider the design of the software during its assessment of how the machine might produce the desired results and what failure modes existed. →reliability modeling and risk management
  - The system noticed that something was wrong and halted the X-ray beam, but merely displayed the word "MALFUNCTION" followed by a number from 1 to 64. The user manual did not explain or even address the error codes, so the operator pressed the P key to override the warning and proceed anyway.
  - AECL personnel and machine operators, initially did not believe complaints. This was likely due to overconfidence.
  - AECL had never tested the Therac-25 with the combination of software and hardware until it was assembled at the hospital.

Therac-25: Engineering Issues

- The failure occurred only when a particular nonstandard sequence of keystrokes was entered on the terminal. This sequence of keystrokes was improbable, and so the problem did not occur often and went unnoticed for a long time.

- The design did not have any hardware interlocks to prevent the electron-beam from operating in its high-energy mode without the target in place.

- The engineer had reused software from older models →cargo cult programming where there is blind reliance on previously created code that is poorly understood and may or may not be applicable.
  - Hardware interlocks masked the software defects in old models - no way of reporting that they had been triggered, so there was no indication of the existence of faulty software commands.

- The hardware provided no way for the software to verify that sensors were working correctly.

- Setting a (one-byte) Boolean variable to “true” was done through the “x=x+1” command, so pressing the “Set” button would result in the system failing to identify the message about incorrect turntable position 1 time out of 256.

Therac 25: Aftermath

- The manufacturer said that the hardware and software had been tested over many years. However, the investigation found that a minimum amount of tests had been run on a simulator, while most of the effort had been directed at the integrated system test.
  - It means that the developers neglected unit testing and did integration testing only.
- The software was not evaluated by independent testers, which may have helped combat cultural biases within the organization.
- Race conditions: The software consisted of several routines running concurrently which did not properly synchronize with each other, so that race conditions occurred. This was missed during testing, since it took some practice before operators were able to work quickly enough to trigger this failure mode.
  - This bug had actually always been present in the older models.
  - A naive assumption is often made that reusing software or using commercial off-the-shelf software increases safety because the software has been exercised extensively. Reusing software modules does not guarantee safety in the new system to which they are transferred due to the development specifics of that system. Rewriting the entire software may be safer in many cases.

Therac-25: Takeaways

- Although most of us won't work on safety-critical systems, software errors can still have a significant impact on our users.
- Usability can sometimes get in the way of safety.
- In safety-critical systems, code should be subject to formal analysis from independent parties from those who developed it. Some level of automated testing at the unit level is also needed, rather than only testing the system as a whole.
- Test, test, test!

Project Quality Management Processes

- Quality Planning Process
  - Quality objectives
- Quality Assurance Process
  - Define how to achieve each objective
- Quality Control Process
  - Monitor and measure performance

Quality Planning

Determine relevant quality standards for the project and what you can do to satisfy them. The inputs to this process are:

- Environmental factors such as agency regulations, rules, standards and guidelines
- Organizational assets such as quality policies, procedures and guidelines, historical data and lessons learned
- Project Overview Statement
- Project Management Plan

A Quality Plan documents:

- How the quality policies will be met
- The metrics that will be used to measure quality
- A process improvement program

Quality Planning Tools and Techniques

- Benefit/cost analysis – less rework, higher productivity, lower costs, increased stakeholder satisfaction.
- Benchmarking
- Flowcharting
  - Cause and effect diagrams
  - System or process flowcharts
- Design of experiments
- Cost of quality – cost of all efforts to achieve quality

Quality Planning Outputs

- Quality management plan
  - Shows how to implement quality policy
- Operational definitions
  - What something is and how it is measured by the quality control process
- Checklists

Quality Assurance

- Apply quality activities so that the project employs the processes needed to assure that quality requirements are met. These can include:
  - Quality Audits
    - Objective is to provide lessons learned
  - Process Analysis
  - Project Quality Management Tools

Software Quality Assurance Techniques

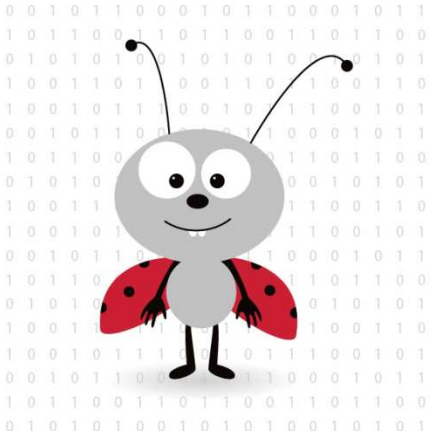
- **Auditing** to determine if the set of standard processes were followed or not.
- **Reviewing** the software product is examined by both the internal and external stakeholders to seek their comments and approval.
- **Code Inspection:** Static testing to find bugs and avoid defect growth in the later stages. It is done by a trained mediator/peer and is based on rules, checklist, entry and exit criteria. The reviewer should not be the author of the code.
- **Design Inspection:** Design inspection is done using a checklist that inspects the below areas of software design:
  - General requirements and design
  - Functional and Interface specifications
  - Requirement traceability
  - Structures and interfaces
  - Logic
  - Performance
  - Error handling and recovery
  - Testability, extensibility
- **Simulation** to model the real-life situation in order to virtually examine the behavior of the system under study.
- **Functional Testing:** Black box testing done to verify what the system does (system specs/features) without considering how it does.
- **Standardization**
- **Static Analysis** done by an automated tool without actually executing the program. E.g.: Software metrics, reverse engineering.
- **Walkthroughs:** A kind of peer review where the developer guides the members of the development team to go through the product and raise queries, suggest alternatives, make comments regarding possible errors, standard violations or any other issues.
- **Path Testing:** A white box testing technique where the complete branch coverage is ensured by executing each independent path at least once.
- **Stress Testing** to check how robust a system is by testing it under heavy load i.e. beyond normal conditions.
- **Six Sigma:** Aims at nearly perfect products or services by process improvement so that the produced software is 99.76 % defect free.

Software Quality Assurance Standards

- **ISO 9000**: This standard is based on seven quality management principles which help the organizations to ensure that their products or services are aligned with the customer needs'.
- **ISO/IEC 25010:2011**: Software engineering — Product quality is an international standard for the evaluation of software quality.
- **Software Process Improvement and Capability Determination** (**SPICE**, also termed ISO/IEC 15504 Information technology – Process assessment): one of the joint ISO and International Electrotechnical Commission (IEC) set of standards documents for the computer software development process and related business management functions.
- **IEEE Standard for Software Quality Assurance Processes**. doi:10.1109/IEEESTD.2014.6835311
- **CMMI level**: (Capability maturity model Integration) Originated in software engineering this model can be used to direct process improvement throughout a project, department, or an organization.
- **Test Maturity Model integration** (**TMMi**): Based on CMMi, this model focuses on maturity levels in software quality management and testing.

Quality Control

- Monitor project performance to determine compliance to quality standards and how to eliminate non-compliance. This will be accomplished by using project management tools, templates and processes.



Quality Control Tools and Techniques

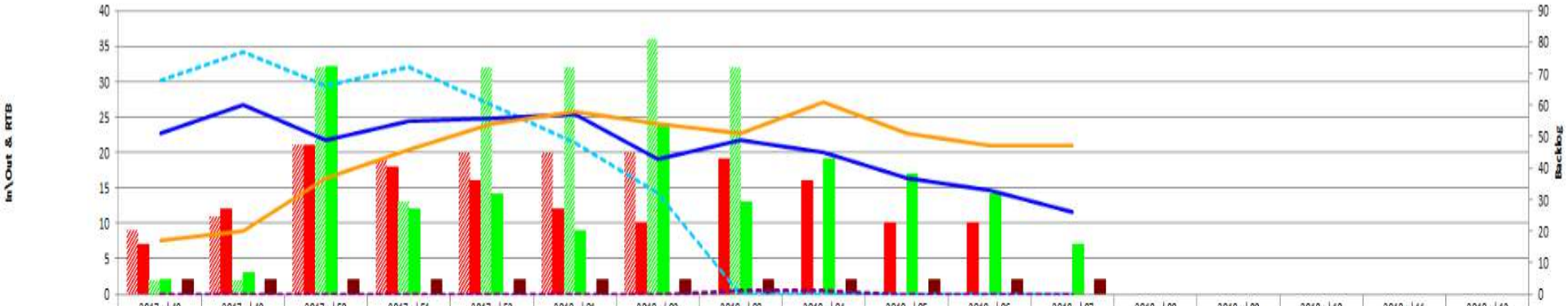
- Inspection
  - Measuring, examining, and testing done to determine conformance to requirements
- Control charts
  - Graphic display of results over time
- Pareto diagrams
  - Histogram ordered by frequency of occurrence showing results per type of cause
- Statistical sampling
- Flowcharting
- Trend analysis
  - Using mathematical techniques to forecast future outcomes based on historical results.

Quality Control Outputs

- Quality improvement
- Acceptance decisions
- Rework
  - Action taken to bring a defective item into compliance with requirements.
  - A frequent cause of project overruns
- Completed checklists
- Process adjustments
  - Immediate corrective and preventive action as a result of quality control measurements

Defect management plan example 1

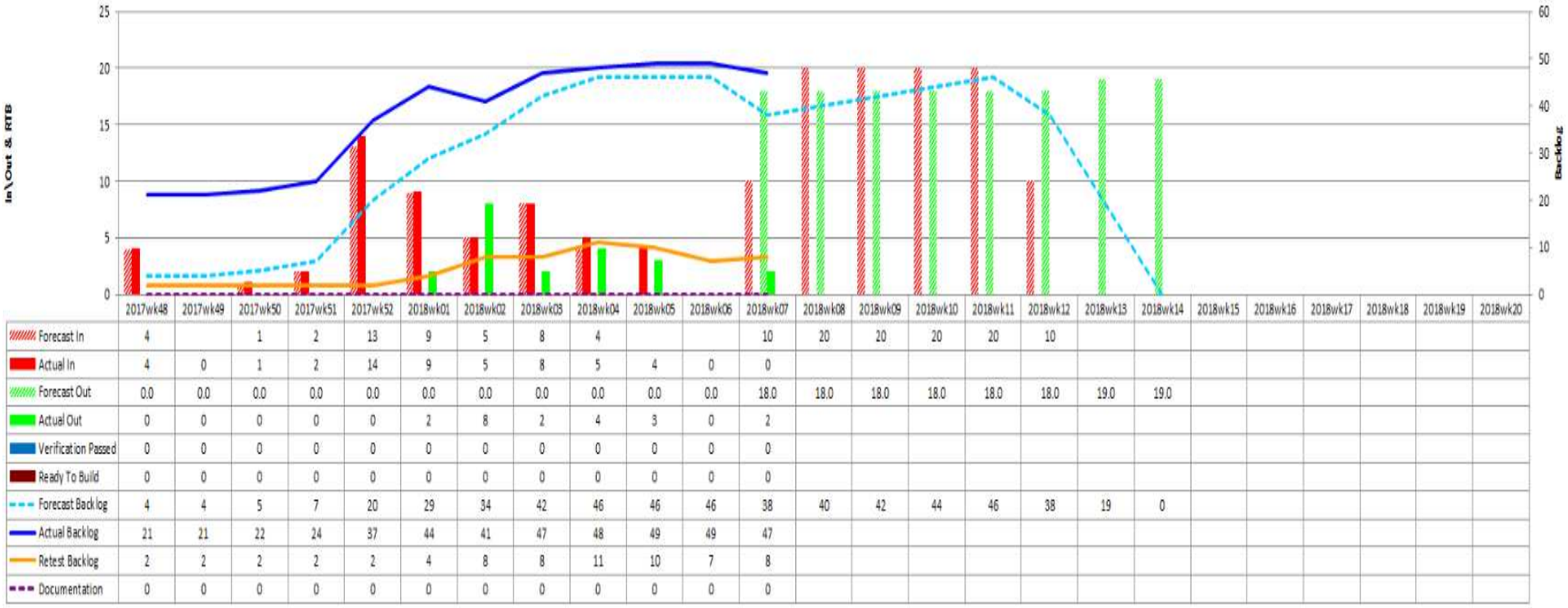
Defect management plan profile for a release milestone - 1



	2017wk48	2017wk49	2017wk50	2017wk51	2017wk52	2018wk01	2018wk02	2018wk03	2018wk04	2018wk05	2018wk06	2018wk07	2018wk08	2018wk09	2018wk10	2018wk11	2018wk12
Forecast In	9	11	21	19	20	20	20										
Actual In	7	12	21	18	16	12	10	19	16	10	10	0					
Forecast Out	2.0	2.0	32.0	13.0	32.0	32.0	36.0	32.0	0.0	0.0	0.0	0.0					
Actual Out	2	3	32	12	14	9	24	13	19	17	14	7					
Verification Passed	0	0	0	0	0	0	0	0	0	0	0	0					
Ready To Build	2	2	2	2	2	2	2	2	2	2	2	2					
Forecast Backlog	68	77	66	72	60	48	32	0	0	0	0	0					
Actual Backlog	51	60	49	55	56	57	43	49	45	37	33	26					
Retest Backlog	17	20	37	46	54	58	54	51	61	51	47	47					
Documentation	0	0	0	0	0	0	0	1	1	0	0	0					

Defect management plan example 2

Defect management plan profile for a release milestone - 2



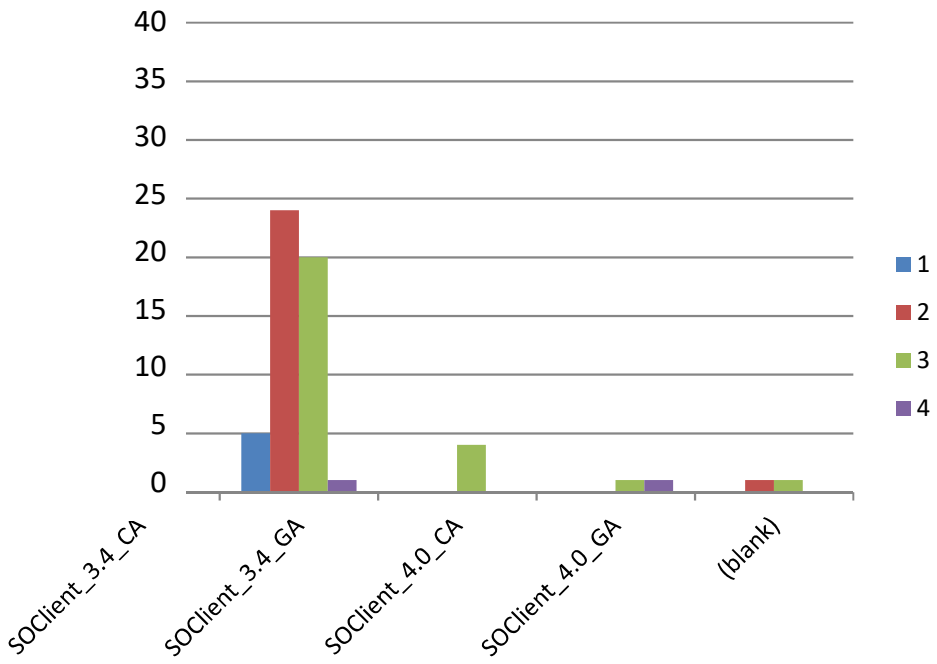
	2017wk48	2017wk49	2017wk50	2017wk51	2017wk52	2018wk01	2018wk02	2018wk03	2018wk04	2018wk05	2018wk06	2018wk07	2018wk08	2018wk09	2018wk10	2018wk11	2018wk12	2018wk13	2018wk14	2018wk15	2018wk16	2018wk17	2018wk18	2018wk19	2018wk20
Forecast In	4		1	2	13	9	5	8	4			10	20	20	20	20	10								
Actual In	4	0	1	2	14	9	5	8	5	4	0	0													
Forecast Out	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	18.0	18.0	18.0	18.0	18.0	18.0	19.0	19.0						
Actual Out	0	0	0	0	0	2	8	2	4	3	0	2													
Verification Passed	0	0	0	0	0	0	0	0	0	0	0	0													
Ready To Build	0	0	0	0	0	0	0	0	0	0	0	0													
Forecast Backlog	4	4	5	7	20	29	34	42	46	46	46	38	40	42	44	46	38	19	0						
Actual Backlog	21	21	22	24	37	44	41	47	48	49	49	47													
Retest Backlog	2	2	2	2	2	4	8	8	11	10	7	8													
Documentation	0	0	0	0	0	0	0	0	0	0	0	0													

Defect Prioritization

- During integration and system verification testing, the defect reports received need to be prioritized.
- Use project milestones as the first deadline for fix
- For further resolution, use priorities to get extra granularity
  - E.g. A defect gating Release Cut milestone with Priority 1 should be fixed immediately. A P3 defect can be fixed later or can be delayed to later milestones based on consensus.
- Defect prioritization should be based on a consensus
  - Periodic triage meetings: "**triage**"
  - Representatives from design, test, project owner, customer, field support should be present.

Defect Prioritization Sample

- Triage of defect reports can be based on following criteria:



Priority	Defect Type
1	Application is not operable or significantly impacted with no feasible work-around E.g. Login/Logout/Password issues, Crashes
2	Application feature or part of a feature is not operable or significantly impacted with no reasonable work-around E.g. Presence not working
3	Application is operable, but some edge use-cases are not working as expected. Work-around is available. E.g. UI refreshes with back/forth operations
4	Ease-of-Use: Usability improvements

Measuring Software Quality

- White box / black box testing
- QA reports (written & executed tests)
- Security response (pentest, nessus scan)
- Production related (active days, code churn)
- CPU usage, memory consumption, download size, etc.

Common Software Quality Metric Definitions

- Agile metrics help to reflect on the development process and improve it:
  - Lead Time: time period starting when the task or project is requested and ending with the completion of the assignment
  - Cycle Time: time from when the task is started and finished (doesn't include wait in queue)
  - Velocity: how quickly individual developers can finish a given task

Common Software Quality Metric Definitions

- QA (Quality Assurance) metrics primarily describe the quality of code and its potential to result in defects or difficult maintenance. Project managers use QA metrics to help predict, prevent, and resolve problems.
  - Test Coverage: how thoroughly testing probes a project (unit tests, user tests, functionality tests)
  - Code Coverage: measures the amount of the codebase that is tested by unit tests
  - Code Complexity Metrics
  - Mean Time to Detect and Mean Time to Repair

Common Software Quality Metric Definitions

- Production metrics reveal how developers use their time, their efficiency, and how often they need to revise their code, and it can help to evaluate a developer's skill and assign them to suitable tasks to increase efficiency.
  - **Active Days**: measures the number of days developers spend actively coding (doesn't include time spent planning)
  - **Productivity**: typically based on the volume of code produced by a developer
  - **Code Churn**: the amount of code that is changed in a piece of software

Sample Metric Definitions

- BHCA = Busy Hour Call Attempt
  - Used to measure network/system capacity
  - E.g. 1M BHCA traffic capacity
  - Failed calls per call attempts

- Software Defect Rate (SDR) calculated per Six Sigma: number of bug reports per every 1000 line of code. Different min & max values at each project phase:
  - Function tests: SDR = 5.0 – 8.0
  - System verification: SDR = 1.0 – 2.0
  - Field reports: SDR= max 0,2

Key Exit Metrics Example

Metrics	Target	Actual	Description / Notes
Tier 3 TC Exec Rate	**100%**		Tier 3: "nice to have" test cases defined to demonstrate functionality that will generally improve the functioning, robustness or breadth of the capability – should be there, but will not directly impair the business or sales ability if not present or fully functional.
Tier 3 TC APR Rate	**95%**		
Tier 2 TC Exec Rate	**100%**		Tier 2: "should have". Test cases defined to demonstrate functionality for business important capabilities – business impairment might occur if the functionality is not delivered, or is not delivered to a high level of quality
Tier 2 TC APR Rate	**98%**		
Tier 1 TC Exec Rate	**100%**		Tier 1: critical / must-have test cases defined to demonstrate functionality for business critical capabilities – if the business is impaired by not having this capability delivered, or by not having it functioning as specified, then it is Tier 1 functionality and test scenario
Tier 1 TC APR Rate	**100%**		
OPEN GA Gating Defects without an accepted resolution plan.	**0**		
OPEN GA Gating Defects	**0-10**		
DSC Sanity Execution	**100%**		
DSC Sanity APR Rate	**99.5%**		

Metric Descriptions Example

Metric	How it is Obtained
% Executed	= Number of Testcases Executed/ Total Number of Testcases
% Pass *(see notes)*	= Number of Testcases Passed / Number of Testcases Executed
% Adjusted Pass Rate	= (Number of Testcases Passed + Number of Testcases Failed against Non-Gating CRs + Number of Testcase failed against CRs in a "Fixed" state)/ Number of Testcases Executed
CR to Testcase Ratio	= Valid CRs written / TCs executed
% of Valid CRs	= (Total # of SV CRs – (# of Invalid CRs from SV))/Total # of SV CRs
Defect / KSLOC ratio	= # of Valid CRs/ KSLOC churn for release
Defect Removal Effeciency	= # of Valid CRs found in SV / Total Number of Valid CRs found post DP3

Project Scheduling

For just about any kind of project, you can start constructing a schedule by asking yourself the following questions:

- What work must be done to achieve the project's objectives?
- What tangible results must your project produce—and when?
- How does each task depend upon the completion of other tasks?
- Who's going to do the work?
- How long do they have to get it done?

Project Scheduling - Work Items

What Work Must Be Done?

- The foundation for any schedule is the work that will achieve the project's objectives and deliver the desired results. Before you can do anything else, you need a list of the tasks to perform, from beginning the project to end.

→Work Breakdown Structure

Project Scheduling – Results and Timing

What Results Must Your Project Produce—and When?

- Monitor and control the progress
  - The earlier you discover that the project is falling behind, the more easily you can take corrective action before it's too late—or too costly.
- Stakeholders expect regular status updates.
- Setting up regular checkpoints, appropriately called milestones, gives you a chance to gauge your progress frequently.
- Every project has a major deliverable at the end but most projects include additional deliverables along the way.
  - Use milestones to identify deliverables and when they're due.

Milestones

- Don’t want sagging projects!

Project Scheduling - Dependencies

How Does Each Task Depend on Other Tasks?

- Some work simply has to be completed before other work can start.
- A relationship between two tasks is known as a task dependency or task link.
- Defining the dependencies between tasks helps you determine which tasks start when, as well as when the project might finish.

Project Scheduling – Resources

Who's Going to Do the Work?

- It's the people and their availability that ultimately controls when tasks get done and how long the project takes.
- Resources are any people, equipment, or materials needed to complete tasks in your project.
- Even if you don't know resource names, you probably know what skills are required to do the work.
  - use a person's name if you have a lucky team member lined up, or
  - fill in generic names when all you know is the type of work or skill required.
- Assign resources to tasks

Project Scheduling – Duration

How Long Will All These Tasks Take?

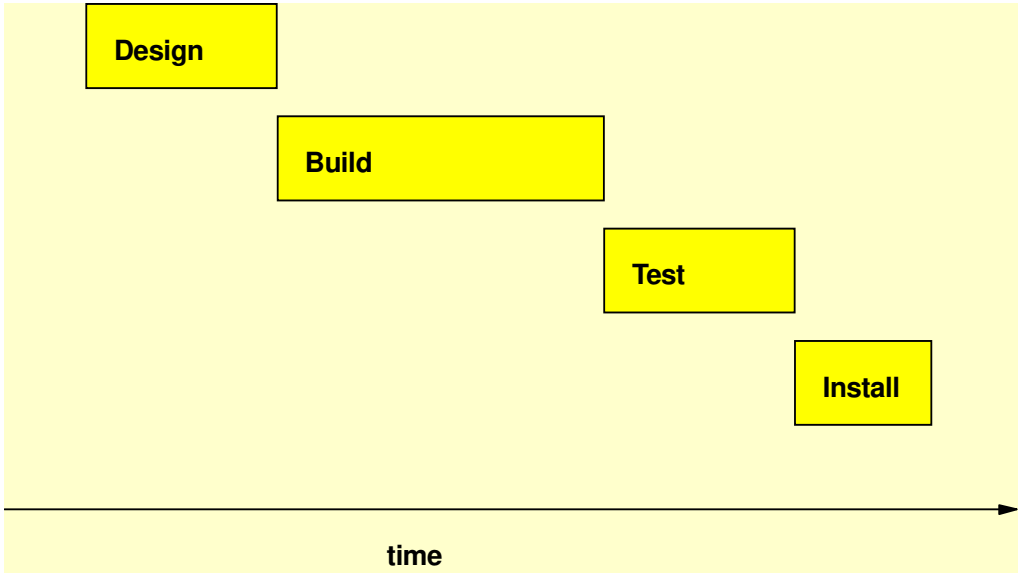
- Use estimates for effort and duration
- Align these based on dependencies
- Find overall project duration

Constructing the Project Network Diagram

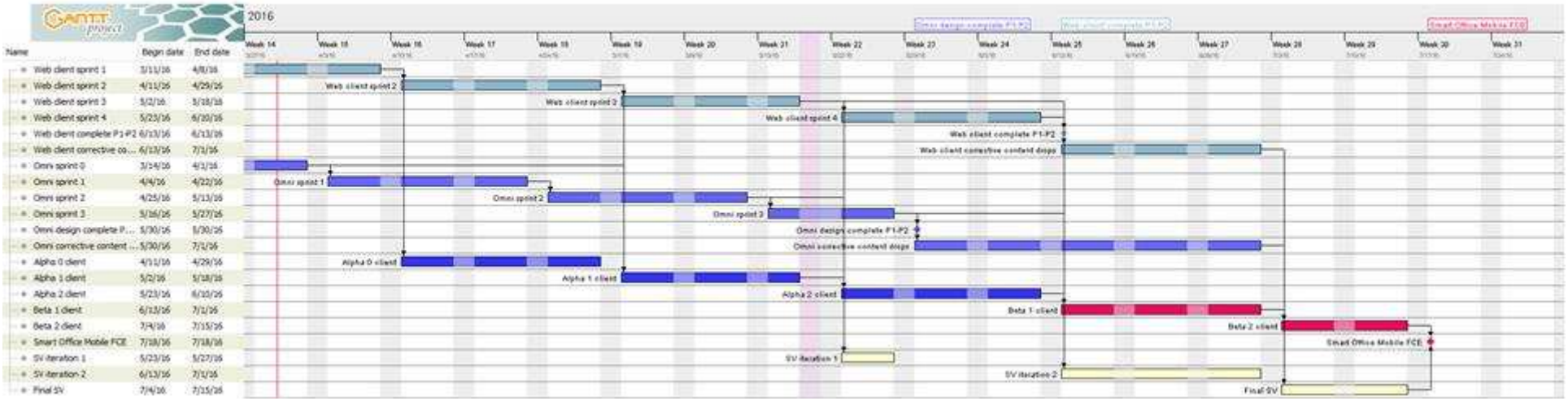
- A *project network diagram* is a pictorial representation of the sequence in which the project work can be done.
  - Represents the relationships between the tasks in the project
- Used to determine the order to perform the tasks
  - The earliest time at which work can begin on every task that makes up the project
  - The earliest expected completion date of the project
- A project schedule can be built using either of the following:
  - Gantt chart
  - Network diagram

Network-based Scheduling – Gantt Charts

- Task bars in the timescale show when tasks begin and end
- Drawbacks
  - Does not contain all of the sequencing information that exists
  - Does not tell whether the schedule that results from the chart completes the project in the shortest possible time or uses the resources most effectively



Gantt Chart Example - 1



Gantt Chart Example - 2

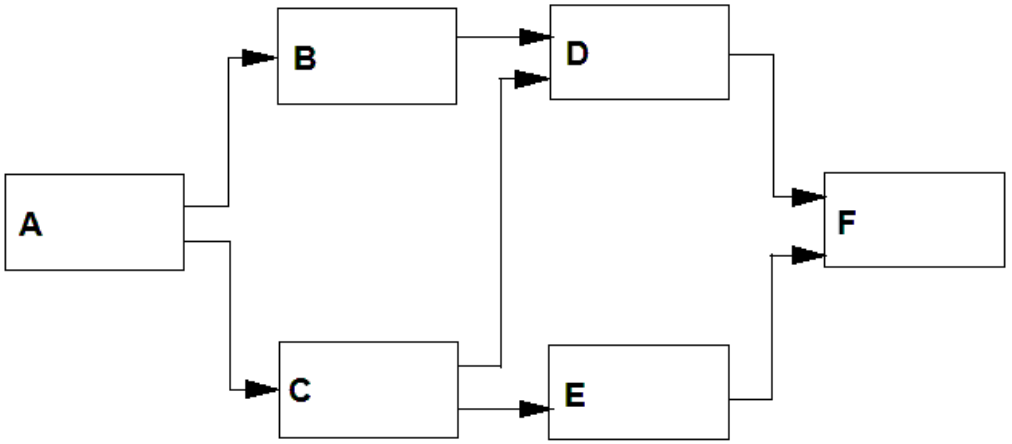
	15-Sep-17	22-Sep-17	29-Sep-17	6-Oct-17	13-Oct-17	20-Oct-17	27-Oct-17	3-Nov-17	10-Nov-17	17-Nov-17	24-Nov-17	1-Dec-17	8-Dec-17	15-Dec-17	22-Dec-17	29-Dec-17	5-Jan-18	12-Jan-18
Schedule - year end delivery target												SDC				DSC CA		
Component Design – MR		3.3 MR2																
Client Design – MR			3.3 MR2			3.3 MR2 Prod												
Component Design – release		3.4 Component Sprint 1						3.4 Component Sprint 2				3.4 Component Sprint 3 - Bug Fix						
Client Design - release				3.4 Client Sprint 1					3.4 Client Sprint 2				3.4 Client Sprint 3 - Bug Fix					
SV testing									SV Sprint 1				SV Sprint 2			Sanity		

Network-based Scheduling – Network Diagrams

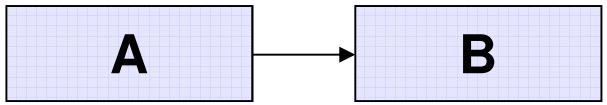
- The network diagram provides a visual layout of the sequence in which project work flows.
- It includes detailed information and serves as an analytical tool for project scheduling and resource management problems.
- It enables you to compute the earliest time at which the project can be completed.
- Network diagrams can be used for
  - Planning
  - Implementation
  - Control

Network-based Scheduling – Task on the Node

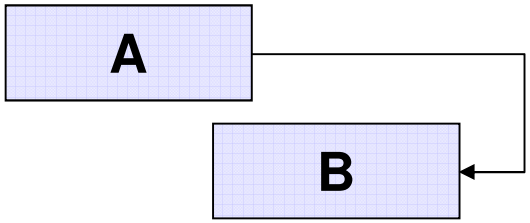
- Precedence diagramming method
- Each task is represented by a rectangle called a task node.
- Arrows represent the predecessor/successor relationships between tasks.
- A dependency is simply a relationship that exists between pairs of tasks.



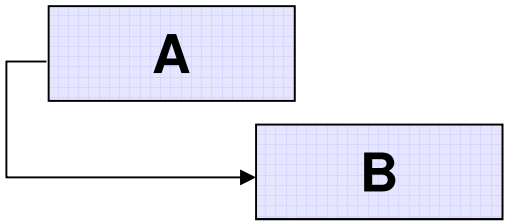
Dependency Relationships



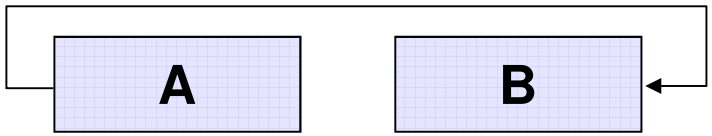
FS: When A finishes, B may start



FF: When A finishes, B may finish



SS: When A starts, B may start



SF: When A starts, B may finish

Dependency Constraints

- The constraints will affect the sequencing of project tasks and, hence, the dependency relations between tasks.

- Technical constraints
  - Discretionary constraints
  - Best practice constraints
  - Logical constraints
  - Unique requirements constraints
- Management constraints
- Inter-project constraints
- Date constraints
  - No earlier than
  - No later than
  - On this date
- Lag variables

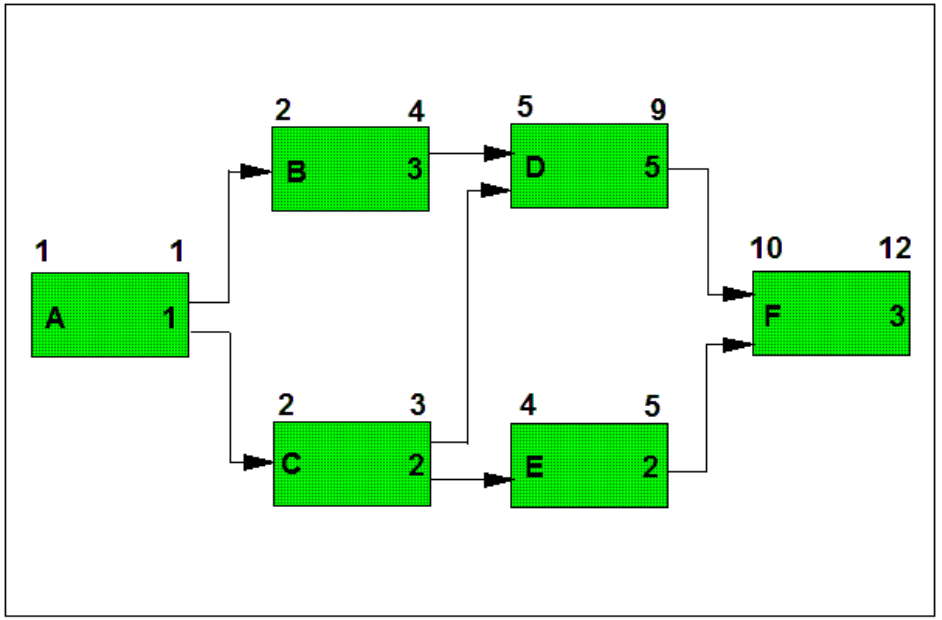
Paths Through the Network

- To establish the project schedule, you need to compute two schedules:
  - The Forward Pass – Early Schedule
    - Consists of the earliest times at which a task can start and finish
  - Backward Pass – Late Schedule
    - Consists of the latest times at which a task can start and finish without delaying the completion date of the project.

- The combination of these two schedules gives:
  - The window of time within which each task must be started and finished in order for the project to be completed on schedule
  - The sequence of tasks that determine the project completion date

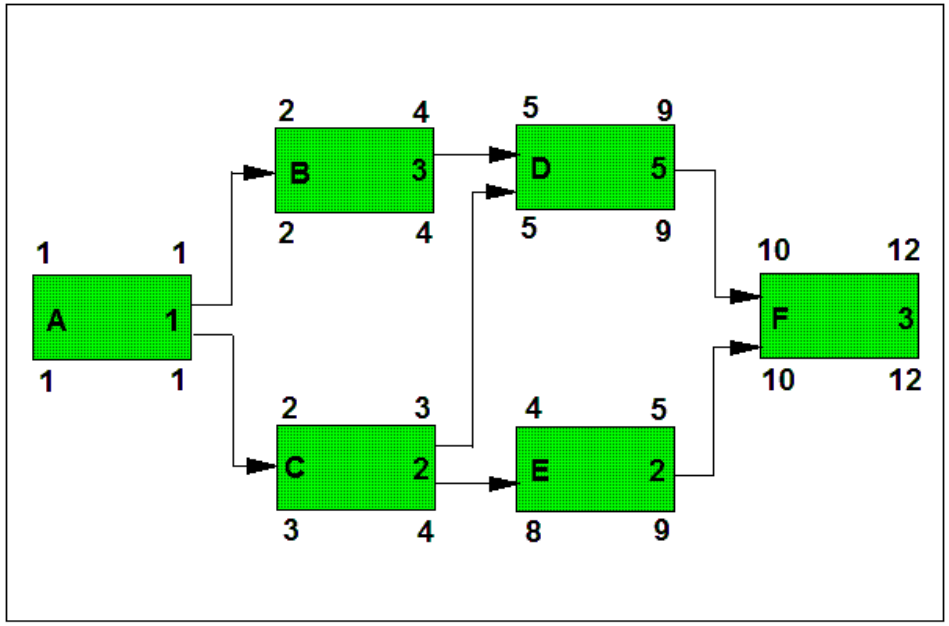
Forward Pass Calculations – Early Schedule

- The Forward Pass – Early Schedule
  - Left to right (start to finish)
  - Determines Early Start and Early Finish
  - ES + duration – 1 = EF



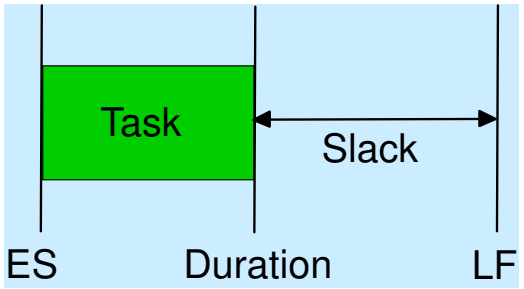
Backward Pass Calculations – Late Schedule

- Backward Pass – Late Schedule
  - Right to left (finish to start)
  - Determines Late Start and Late Finish
  - LF - duration + 1 = LS



Slack (Float) Time

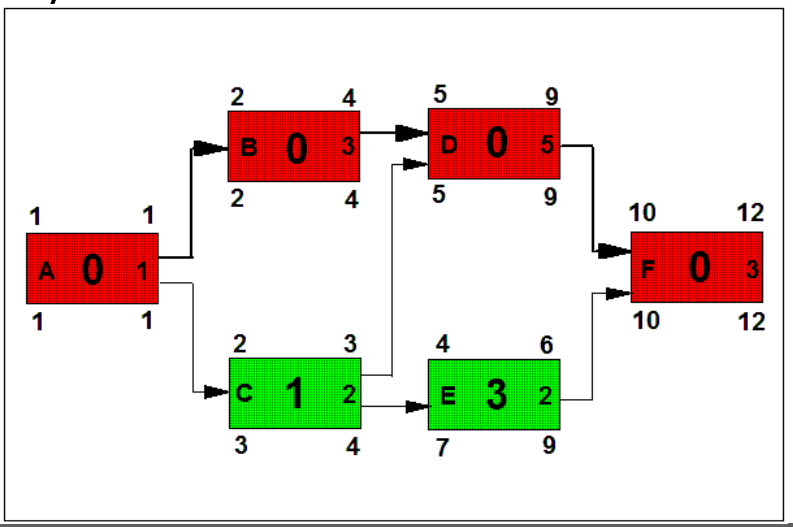
- It is the difference between the late finish and the early finish (LF – EF).
  - Also called float.



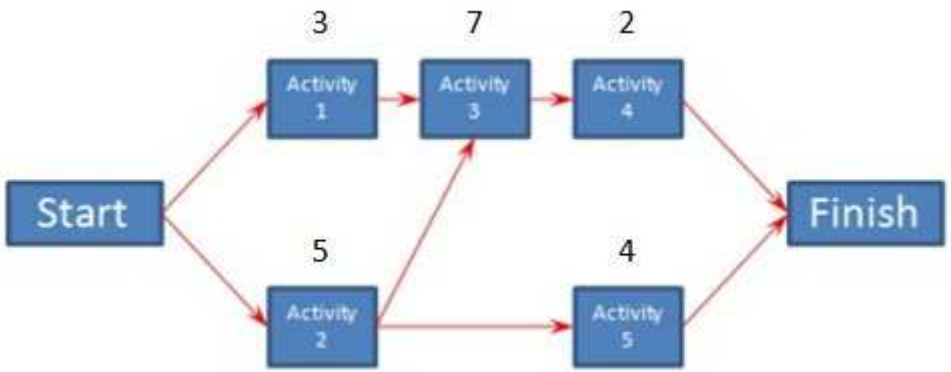
- Total Slack
  - Time that a task can be delayed without impacting the early schedule of the project.
- Free Slack
  - Time a task can be delayed without impacting the early schedule of its successor tasks.

Critical Path

- The sequence of tasks that determine the project completion date.
  - The longest duration path in the network diagram
  - The sequence of tasks whose early schedule and late schedule are the same
  - The sequence of tasks with zero slack (in the general case, with minimum slack)



Critical Path Example



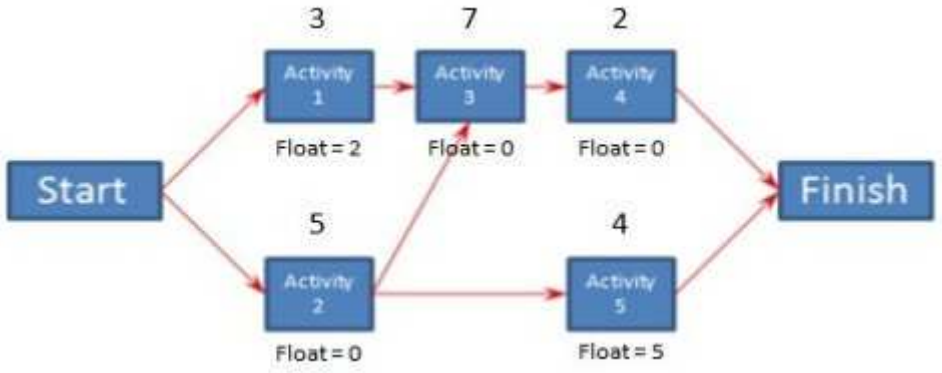
- There are three paths through this project

Start → Activity 1 → Activity 3 → Activity 4 → Finish	3+7+2=12
Start → Activity 2 → Activity 3 → Activity 4 → Finish	5+7+2=14
Start → Activity 2 → Activity 5 → Finish	5+4=9

Float Determination

- Use the Critical Path Method
- Start with the activities on the critical path.
  - Each of those activities has a float of zero. If any of those activities slips, the project will be delayed.
- Then take the next longest path.
- Subtract it's duration from the duration of the critical path.
- That's the float for each of the activities on that path.
- Continue doing the same for each subsequent longest path until each activity’s float has been determined.
- If an activity is on two paths, its float will be based on the longer path that it belongs to.

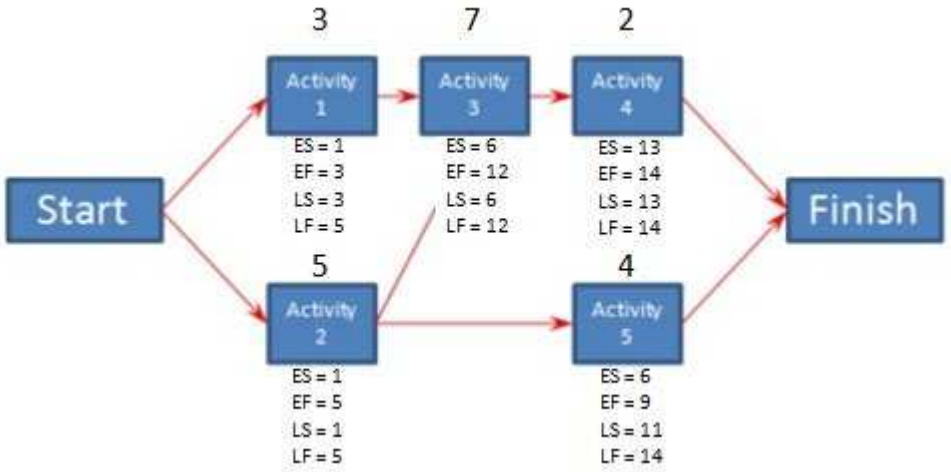
Float Determination Example



- Activities 2, 3, and 4 are on the critical path so they have a float of zero.

- The next longest path is Activities 1, 3, and 4.
  - Since Activities 3 and 4 are also on the critical path, their float will remain as zero.
  - For any remaining activities, in this case Activity 1, the float will be the duration of the critical path minus the duration of this path. 14 - 12 = 2. So Activity 1 has a float of 2.
- The next longest path is Activities 2 and 5.
  - Activity 2 is on the critical path so it will have a float of zero.
  - Activity 5 has a float of 14 - 9, which is 5. So as long as Activity 5 doesn't slip more than 5 days, it won't cause a delay to the project.

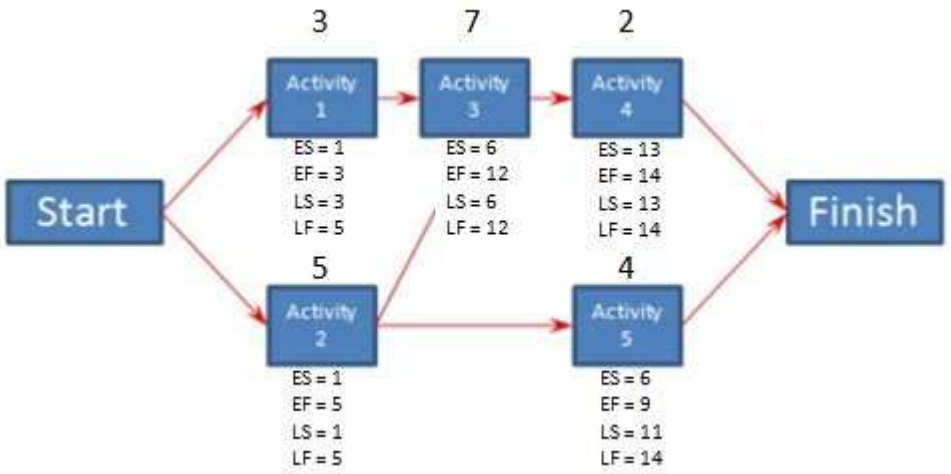
Early Start & Early Finish Calculation Example



Forward Pass

- Starting with the critical path, the Early Start (ES) of the first activity is one.
- ES is the previous activity's EF + 1
- Early Finish (EF) = ES + duration – 1
- If an activity has more than one predecessor, to calculate its ES you will use the activity with the latest EF.
- Activity 2 is the first activity on the critical path:
  - ES = 1, EF = 1 + 5 -1 = 5.
- Move to the next activity in the path, in this case Activity 3.
  - ES = 5 + 1 = 6.
  - EF = 6 + 7 - 1 = 12.

Late Start & Late Finish Calculation Example



Backward Pass

- Start once again with the critical path, but this time begin from the last activity in the path.
- The Late Finish (LF) for the last activity in every path is the same as the last activity's EF in the critical path.
- The Late Start (LS) = LF - duration + 1
- Move on to the next activity in the path. Its LF = previous activity's LS - 1
- Activity 4 is the last activity on the critical path.
  - Its LF is the same as its EF = 14.
  - LS = 14 - 2 + 1 = 13.
- The next Activity in the critical path is Activity 3.
  - LF = Activity 4 LS - 1 = 13 -1 = 12
  - LS = 12 - 7 + 1 = 6

Building the Initial Dependency Diagram

1. Enter task name and duration into software tool
2. Print PERT Diagram
3. Cut out each task node and affix to a Sticky Note
4. Write task ID # on Post-It Note
5. Place Sticky Notes on right side of whiteboard
6. Position start node on left side of whiteboard
7. Move all tasks with no predecessor to left side and connect to start node
8. Move all tasks on the right side with predecessors on the left side to the left side and connect with single line
9. Continue until all task nodes have been moved to left side

Compressing the Schedule

- When project calculations will result in a project completion date beyond the required completion date
- Convert work in series to work in parallel
- Identify strategies for locating potential dependency changes.
  - Focus your attention on critical path tasks
- Focusing on tasks that are partitionable - whose work can be assigned to two or more individuals working in parallel.

Schedule Compression Techniques

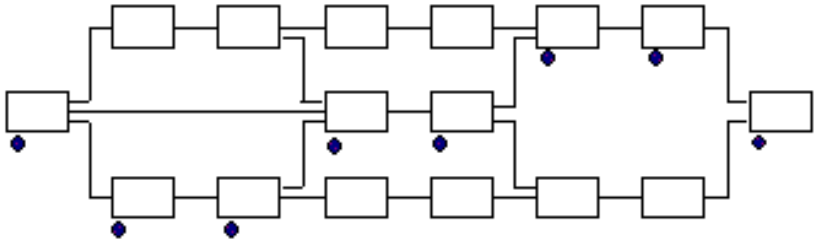
- Replace Finish-to-Start with Start-to-Start dependencies
- Replace a team member with a more skilled person
- Add resources
  - From non-critical path tasks to critical path tasks
  - From other projects
  - To where
    - critical path tasks
    - high-risk tasks
    - tasks with large duration variances

Schedule Compression Iterations

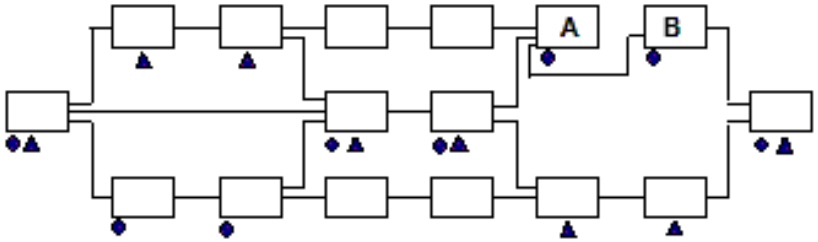
Initial critical path tasks

New critical path tasks

Third critical path tasks

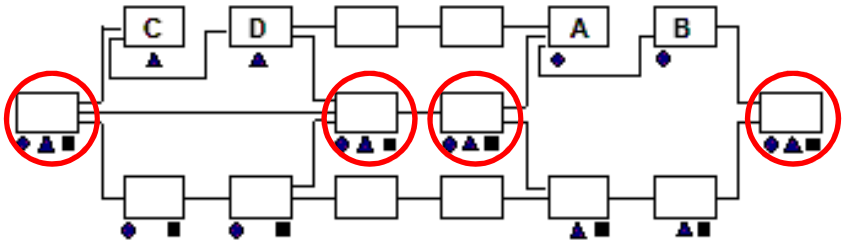


Original critical path



Critical path after changing AB from FS to SS

After applying schedule compression several times, some tasks always remain on the critical path.



Critical path after changing CD from FS to SS

Management Reserve

- The extra day that you add to the estimate to make sure you can get the work done is called padding.
- Parkinson's Law states that work will expand to the time allotted to complete it.
  - DO NOT PAD ACTIVITY DURATION
- Management reserve is a schedule contingency for the unexpected
  - Its size is a percentage of total project work hours
  - Apply as the last activity in the project
  - Make it visible and manage it

Summary

- Quality Management
  - Quality Planning: Determine relevant quality standards for the project and what you can do to satisfy them.
  - Quality Assurance: Apply quality activities so that the project employs the processes needed to assure that quality requirements are met.
  - Quality Control: Monitor project performance to determine compliance to quality standards and how to eliminate non-compliance.
- Project Scheduling
  - Define work items, results and timing, dependencies, resources, and duration
  - Construct the project network diagram to obtain a graphical representation of the sequence in which the project work can be done.
  - Critical path is the sequence of tasks that determine the project completion date.
  - Convert work in series to work in parallel to compress the schedule
  - Parkinson's Law states that work will expand to the time allotted to complete it.
  - Management reserve is a schedule contingency for the unexpected.

Schedule Calculation Example - 1

Project: Assembling a train set

Step 1: Find Activities

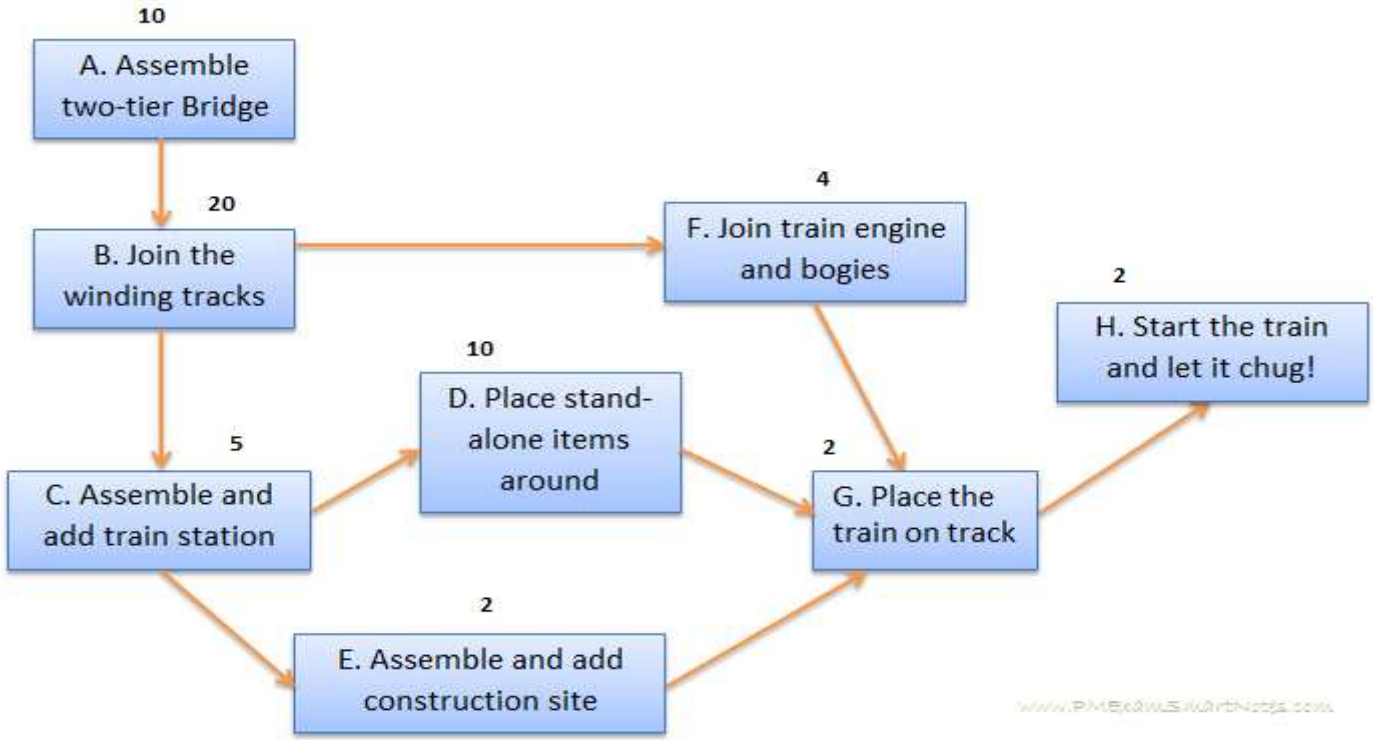
Activities for this project are as below (output from WBS)

- Assemble two-tier bridge
- Join winding tracks
- Assemble and add train station
- Place standalone items around
- Assemble and add construction site
- Join train engine and bogies
- Place the train on the track
- Start the engine and let it chug!

Schedule Calculation Example - 2

Step 2: Build Schedule Network Diagram

- Sequence activities and build schedule network diagram (This is how it looks, with individual activity duration in minutes):

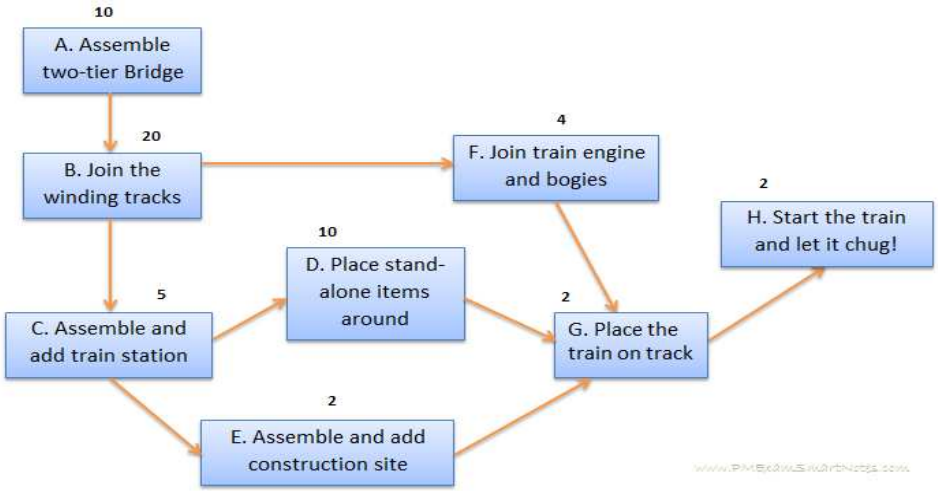


Schedule Calculation Example - 3

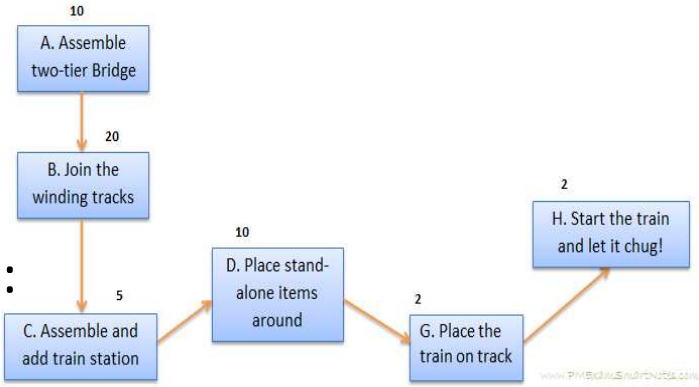
Step 3: Find all Possible Paths

Find all possible paths through the diagram, there are 3 in our case:

- A -> B -> F -> G -> H
- A -> B -> C -> D -> G -> H
- A -> B -> C -> E -> G -> H



Schedule Calculation Example - 4



Step 4: Calculate Duration for Each Path

Let us see the duration for each of these paths:

- A -> B -> F -> G -> H —> 10+20+4+2+2 = 38 minutes
- A -> B -> C -> D -> G -> H —> 10+20+5+10+2+2 = 49 minutes
- A -> B -> C -> E -> G -> H —> 10+20+5+2+2+2= 41 minutes
- Note that sum of durations of all activities on critical path comes 49 minutes, and sum of duration of ALL activities on the project is much longer. If managed well, the whole project can be completed within the critical path's duration.

Schedule Calculation Example - 5

Step 5: Arrange the paths in decreasing order of their total duration, starting with Critical path to calculate float for activities:

- A -> B -> C -> D -> G -> H —> 10+20+5+10+2+2 = 49
- A -> B -> C -> E -> G -> H —> 10+20+5+2+2+2= 41
- A -> B -> F -> G -> H —> 10+20+4+2+2 = 38
- The float for each activity on the critical path is zero.

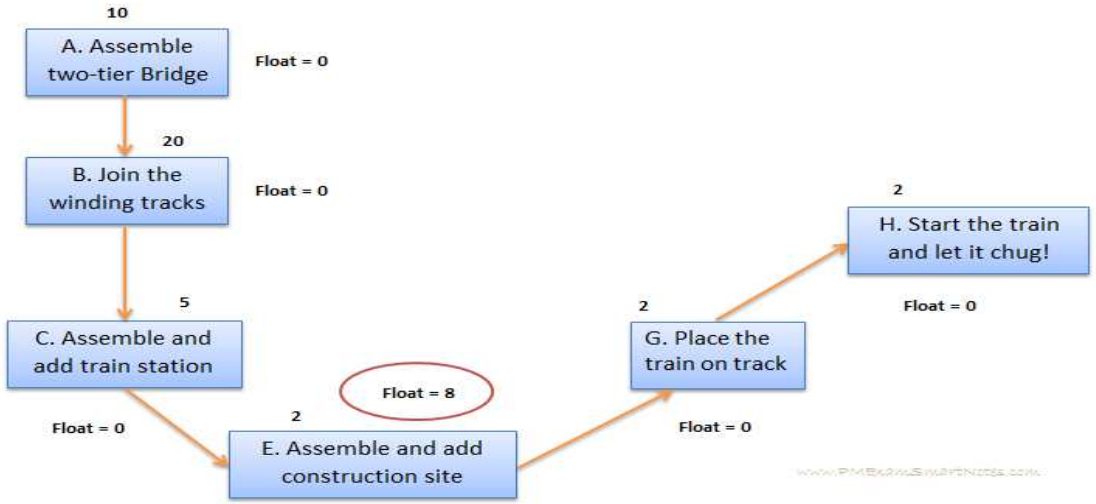
Schedule Calculation Example - 6

Step 6: Find float for activities on the second longest path

- This would be the difference between total duration of critical path and next longest path.

  49-41= 8 minutes

- Assign this to ALL activities on this path, which do not already have a float.



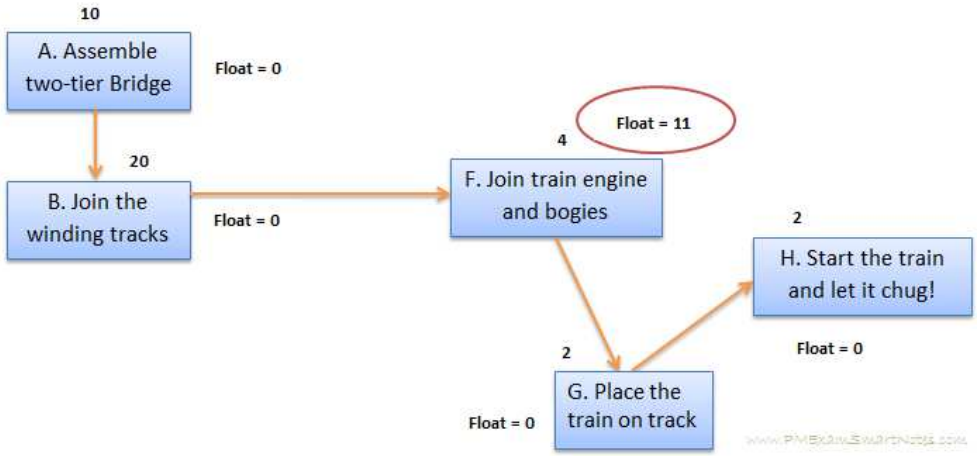
Schedule Calculation Example - 7

Step 7: Do the same to all remaining paths, for unassigned activities

- Calculate difference between critical path and third path's total duration, and assign this to activities on the third path – excluding any which already have a float assigned in previous step.

  49-38 = 11 minutes

- Do this step for rest of the identified paths and you have float for all activities.

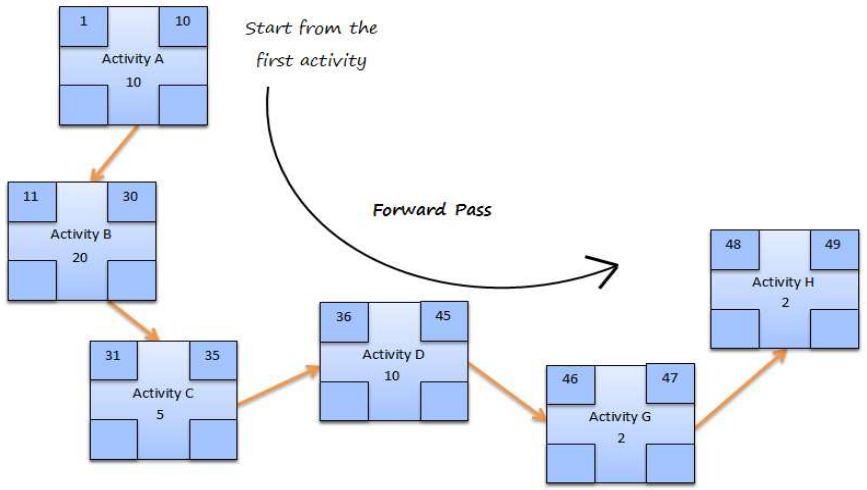


Schedule Calculation Example - 8

Step 8: Calculating Early start and finish (take a FORWARD pass through network path)

- Always start with the critical path and then go with paths with descending order of their total duration.
- Early start of first activity on critical path is always 1. Write it at the top left corner of that activity box.
- Add its activity duration to this early start number and reduce it by one. Write the resulting number on the top right corner of activity box.
- Take the subsequent number of this early finish and write as early start for next activity. Continue this till you reach the end of critical path.
- Select the network path with second highest total duration, and calculate early starts and finishes. If you find an activity with early start and finish already written do not overwrite them. Do the same for remaining network paths.

Early start and early finish for critical path →

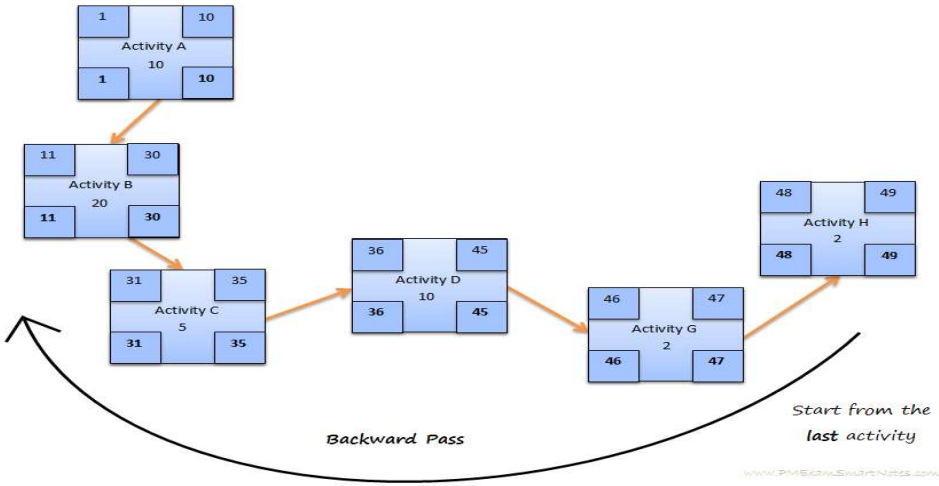


Schedule Calculation Example - 9

Step 9: Calculating Late start and finish (take BACKWARD pass through network path)

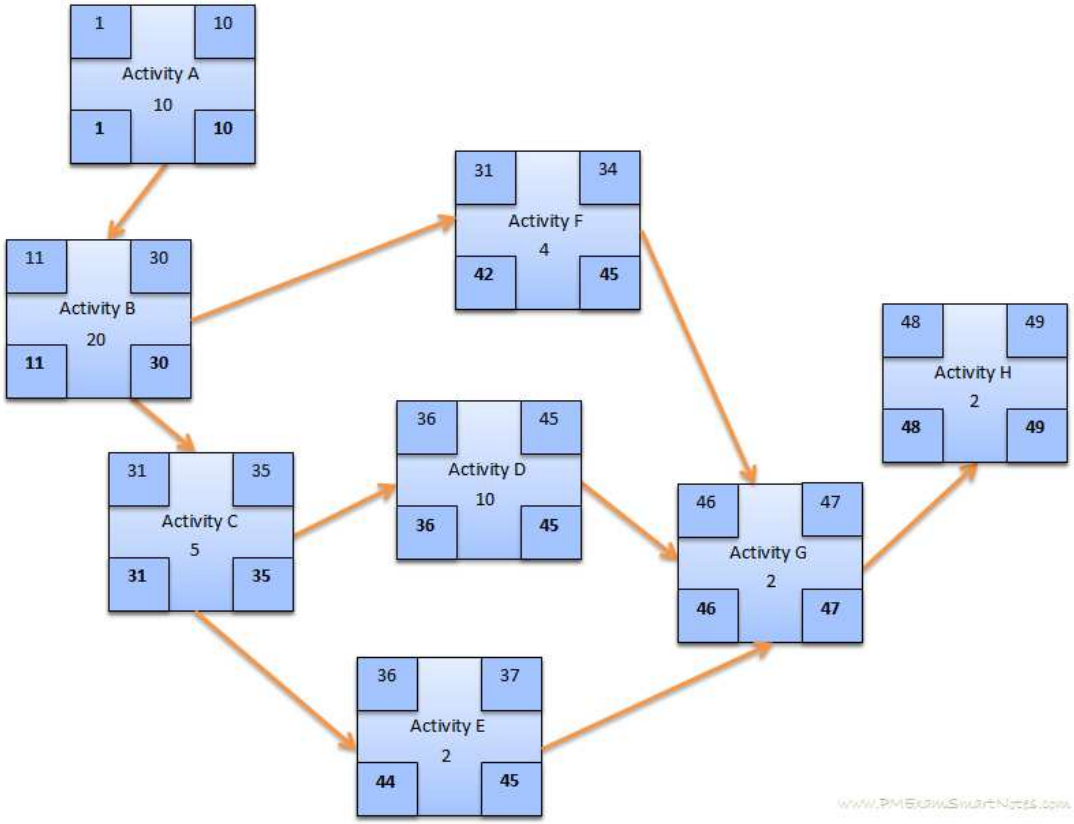
- Start with the critical path, beginning at the last activity's late finish.
- Late finish of last activity on the critical path is same as its early finish. Write this number at the bottom right corner.
- Calculate late start of this activity as the late finish minus activity duration plus 1. This calculation has the same reason – start and finish are both included in the duration. Write this number at the bottom left corner.
- Write this late start of the activity minus 1, as the late finish of previous activity. Continue this way all way till you reach the late start of first activity on the critical path.
- Select the network path with second highest total duration, and write late starts and finishes beginning at the last activity of that path. Do the same for remaining network paths.

Late start, finish for the critical path →



Schedule Calculation Example - 10

- Early start, finish and Late start, finish for the entire schedule network diagram



Program Evaluation and Review Technique (PERT)

- A statistical tool developed by the US Navy in the 1950s which is designed to analyze and represent the tasks involved in completing a given project. It is commonly used in conjunction with the critical path method.
- Used more in projects where time is the major factor rather than cost
- Clearly illustrates task dependencies
- PERT uses Monte Carlo simulations to show possible project outcomes. The result is a probability of a project finishing on time given combinations of optimistic, pessimistic, and expected values.
- A PERT chart presents a graphic illustration of a project as a network diagram consisting of numbered nodes representing events, or milestones in the project linked by labeled vectors representing tasks in the project.

Gantt Chart

- A type of bar chart that illustrates a project schedule.
- Lists the tasks to be performed on the vertical axis, and time intervals on the horizontal axis.
- The width of the horizontal bars in the graph show the duration of each activity.
- Gantt charts illustrate the start and finish dates of the terminal elements and summary elements of a project.

References

- Time – cost calculations:
  http://osp.mans.edu.eg/elbeltagi/CM%20CH8%20Time-Cost.pdf
  https://docplayer.net/12592036-Chapter-8-project-time-cost-trade-off.html

- The Empirical Laws of Software Engineering (for short descriptions of Parkinson's Law, Pareto Principle, Brooks's Law, Murphy's Law and others:
  - https://brainhub.eu/blog/empirical-laws-software-engineering/
  - https://www.timsommer.be/famous-laws-of-software-development/
  - https://exceptionnotfound.net/fundamental-laws-of-software-development/
  - https://daverupert.com/2018/04/eponymous-laws-of-tech/