Project Planning

- Sanghamitra De

Project Planning

- The basic goal of planning is to look into the future, identify the activities that need to be done to complete the project successfully, and plan the scheduling and resources.
- The inputs to the planning activity are the requirements specification and the architecture description.

Project Planning

Major issues addressed by project planning:

1) Process planning
2) Effort estimation
3) Schedule and Resource Estimation
4) Configuration management plans
5) Quality plans
6) Risk management
7) Project monitoring plans

1) Process planning

- Specifies the various stages in the process, the entry criteria for each stage, the exit criteria, and the verification activities that will be done at the end of the stage.
- Requires selecting a standard process and tailoring it for the project.
- After tailoring, the process specification for the project is available.
- Process specification guides rest of the planning, particularly detailed scheduling.

2) Effort estimation

- The primary reason for cost and schedule estimation is cost-benefit analysis, and project monitoring and control, bidding for software projects.
- Most cost estimation procedures focus on estimating effort in terms of person-months (PM).
- For a software development project, effort and schedule estimates are essential prerequisites for managing the project.
- Effort and schedule estimates are required to determine the staffing level for a project during different phases.

Effort Estimation Models

- Basic idea of having a model for estimation is that it reduces the problem of estimation to estimating or determining the value of the "key parameters" that characterize the project, based on which the effort can be estimated.
- Primary factor that controls the effort is the size of the project, that is, the larger the project, the greater the effort requirement

Effort Estimation Models

- One common approach for estimating effort is to make it a function of project size:

$$\textbf{EFFORT} = \textbf{a} * \textbf{SIZE}^{\textbf{b}}$$

Here **a** and **b** are constants, and project size is generally in **KLOC** or **function points**. *Values for* these *constants* for a particular process are determined *through regression analysis*, which is applied to data about the projects that has been performed in the past.

COnstructive **CO**st **MO**del (**COCOMO**)

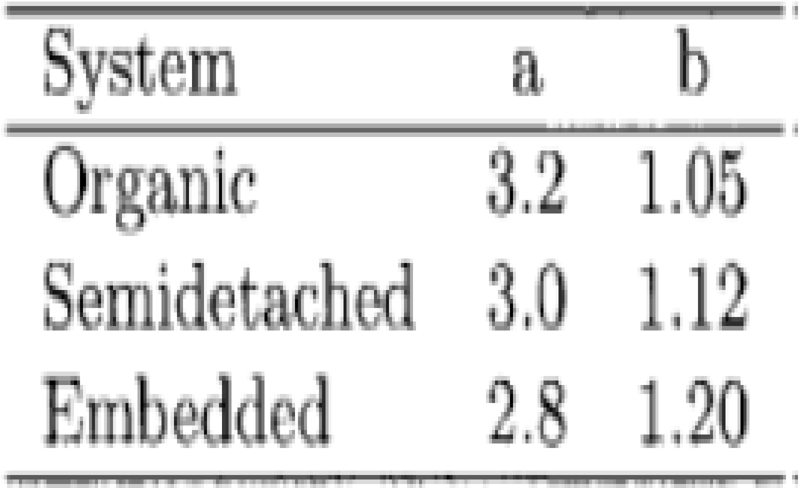
Basic steps:

- Obtain an initial estimate of the development effort from the estimate of thousands of delivered lines of source code (KLOC).
- Determine a set of 15 multiplying factors from different attributes of the project.
- Adjust the effort estimate by multiplying the initial estimate with all the multiplying factors.

COCOMO

$$E_i = a * (KLOC)^b$$

Value of the constants a and b depend on the project type.



System	a	b
Organic	3.2	1.05
Semidetached	3.0	1.12
Embedded	2.8	1.20

COCOMO

- **Organic** projects- those that are relatively straightforward and developed by a small team
- **Embedded** projects- those that are ambitious and novel, with stringent constraints from the environment and high requirements for such aspects as interfacing and reliability

COCOMO

- There are 15 different attributes, called cost driver attributes that determine the multiplying factors.
- These factors depend on product, computer, personnel, and technology attributes (called project attributes).
- Examples of the attributes are: *required software reliability* (RELY), *product complexity* (CPLX), *analyst capability* (ACAP), *application experience* (AEXP), *use of modern tools* (TOOL), and *required development schedule* (SCHD).

COCOMO

- Each cost driver has a rating scale, and for each rating, a multiplying factor is provided.
- The COCOMO approach also provides guidelines for assessing the rating for the different attributes.
- The multiplying factors for all 15 cost drivers are multiplied to get the effort adjustment factor (EAF).
- Final effort estimate, E, is obtained by multiplying the initial estimate by the EAF. By this method, the overall cost of the project can be estimated.

$$\mathbf{E = EAF * E_i}$$

	Rating				
Cost Drivers	Very Low	Low	Nom-inal	High	Very High
Product Attributes					
RELY, required reliability	.75	.88	1.00	1.15	1.40
DATA, database size		.94	1.00	1.08	1.16
CPLX, product complexity	.70	.85	1.00	1.15	1.30
Computer Attributes					
TIME, execution time constraint			1.00	1.11	1.30
STOR, main storage constraint			1.00	1.06	1.21
VITR, virtual machine volatility		.87	1.00	1.15	1.30
TURN, computer turnaround time		.87	1.00	1.07	1.15
Personnel Attributes					
ACAP, analyst capability	1.46	1.19	1.00	.86	.71
AEXP, application exp.	1.29	1.13	1.00	.91	.82
PCAP, programmer capability	1.42	1.17	1.00	.86	.70
VEXP, virtual machine exp.	1.21	1.10	1.00	.90	
LEXP, prog. language exp.	1.14	1.07	1.00	.95	
Project Attributes					
MODP, modern prog. practices	1.24	1.10	1.00	.91	.82
TOOL, use of SW tools	1.24	1.10	1.00	.91	.83
SCHED, development schedule	1.23	1.08	1.00	1.04	1.10

Table : Effort multipliers for different cost drivers.

3) Schedule and Resource Estimation

- Once effort is estimated, various schedules (or project duration) are possible, depending on the number of resources (people) put on the project.
- E.g. for a project whose effort estimate is 56 person-months, a total schedule of 8 months is possible with 7 people. A schedule of 7 months with 8 people is also possible, as is a schedule of approximately 9 months with 6 people.
- **Manpower and months are not fully interchangeable in a software project.**

Schedule and Resource Estimation

- In a project, the scheduling activity can be broken into two sub activities: determining the overall schedule (the project duration) with major milestones, and developing the detailed schedule of the various tasks.
- One method to determine the normal (or nominal) overall schedule is to determine it as a function of effort.
- Any such function has to be determined from data from completed projects using statistical techniques like fitting a regression curve through the scatter plot obtained by plotting the effort and schedule of past projects.
- This curve is generally nonlinear because the **schedule does not grow linearly with effort**.
- In COCOMO, the equation for schedule for an organic type of software is:

$$\mathbf{M = 2.5 * E^{0.38}}$$

- It should be clear that **schedule is not a function solely of effort.**

Schedule and Resource Estimation

- The number of people in a software project tends to follow the Rayleigh curve.
- That is, in the beginning and the end, few people work on the project; the peak team size (PTS) is reached somewhere near the middle of the project.
- This behavior occurs because only a *few people* are needed in the *initial phases* of requirements analysis and design. The human resources requirement *peaks during coding and unit testing*. Again, during system *testing and integration*, *fewer people* are required.

Schedule and Resource Estimation

- Generally speaking, design requires about a quarter of the schedule, build consumes about half, and integration and system testing consume the remaining quarter.
- COCOMO gives 19% for design, 62% for programming, and 18% for integration

Schedule and Resource Estimation

- Once the milestones and the resources are fixed, it is time to set the detailed scheduling.
- For detailed schedules, the major tasks fixed while planning the milestones are broken into small schedulable activities in a hierarchical manner.
- For each detailed task, the project manager estimates the time required to complete it and assigns a suitable resource so that the overall schedule is met.
- At each level of refinement, the project manager determines the effort for the overall task from the detailed schedule and checks it against the effort estimates.
- If this detailed schedule is not consistent with the overall schedule and effort estimates, the detailed schedule must be changed.
- Once the overall schedule is fixed, detailing for a phase may only be done at the start of that phase.

Schedule and Resource Estimation

- For each lowest-level activity, the project manager specifies the effort, duration, start date, end date, and resources.
- Dependencies between activities, due either to an inherent dependency or to a resource-related dependency may also be specified.
- From specific tools the overall effort and schedule of higher level tasks can be determined.
- A detailed project schedule is never static. Changes may be needed because the actual progress in the project may be different from what was planned, because newer tasks are added in response to change requests, or because of other unforeseen situations.
- The final schedule, frequently maintained using some suitable tool, is often the most "live" project plan document.
- During the project, if plans are changed and additional activities are done, then the changes must be reflected in the detailed schedule, as this reflects the tasks actually planned to be performed.

Staffing and Team Structure

- Number of resources is fixed when schedule is being planned.
- Detailed scheduling is done only after actual assignment of people has been done, as task assignment needs information about the capabilities of the team members.
- Project's team is led by a project manager.
- This form of hierarchical team organization was earlier called the **Chief Programmer Team**.
- Project manager is responsible for planning and task assignment. All major technical decisions of the project. He does most of the design and assigns coding of the different parts of the design to the programmers.
- Team typically consists of *programmers*, *testers*, *a configuration controller*, and possibly a *librarian* for documentation.
- Other possible roles: database manager, network manager, backup project manager, or a backup configuration controller. Since all are logical roles, one person may do multiple such roles.

Staffing and Team Structure

- A different team organization is the **egoless team**.
- Egoless teams consist of ten or fewer programmers.
- Goals of the group are set by consensus, and input from every member is taken for major decisions.
- Group leadership rotates among the group members.
- Due to their nature, egoless teams are sometimes called democratic teams.
- This structure allows input from all members, which can lead to better decisions for difficult problems.
- Well suited for long-term research-type projects that do not have time constraints.
- It is not suitable for regular tasks that have time constraints; for such tasks, the communication in democratic structure is unnecessary and results in inefficiency.

Staffing and Team Structure

- For very large product developments, another structure is used.
- This structure recognizes that there are three main task categories in software development—management related, development related, and testing related.
- It also recognizes that it is often desirable to have the test and development team be relatively independent, and also not to have the developers or testers report to a nontechnical manager.
- In this structure, there is an **overall unit manager**, under whom there are three small hierarchic organizations—for program management, for development, and for testing.
- The primary job of developers is to write code and they work under a **development manager**.
- The responsibility of the testers is to test the code and they work under a **test manager**.
- The **program manager** provides the specifications for what is being built, and ensure that development and testing are properly coordinated.
- In a large product this structure may be replicated, one for each major unit. This type of team organization is used in corporations like Microsoft.

4) Configuration management plans

- SCM plan, has to identify the activities that must be performed, give guidelines for performing the activities, and allocate resources for them.
- Planning for configuration management involves identifying the configuration items and specifying the procedures to be used for controlling and implementing changes to them.
- The configuration controller does the CM planning when the project has been initiated and the operating environment and requirements specifications are known.
- Output of this phase is the CM plan

Configuration management plans

Activities in this stage include:

a) Identify configuration items, including customer-supplied and purchased items.
b) Define a naming scheme for configuration items.
c) Define the directory structure needed for CM.
d) Define version management procedures, and methods for tracking changes to configuration items.
e) Define access restrictions.
f) Define change control procedures.
g) Identify and define the responsibility of the CC.
h) Identify points at which baselines will be created.
i) Define a backup procedure and a reconciliation procedure, if needed.
j) Define a release procedure.

5) Quality plans

- Software quality can be defined as: "delivered defect density”.
- Defect is something in software that causes the software to behave in a manner that is inconsistent with the requirements or needs of the customer. Defect removal will result in some change being made to the software.
- To ensure that the final product is of high quality, some quality control (QC) activities must be performed throughout the development.
- A QC task is one whose main purpose is to identify defects.
- The purpose of a quality plan in a project is to specify the activities that need to be performed for identifying and removing defects, and the tools and methods that may be used for that purpose.
- The quality plan specifies the tasks that need to be undertaken at different times in the project to improve the software quality by removing defects, and how they are to be managed.
- To ensure that the delivered software is of good quality, it is essential to ensure that all work products are also of good quality. For this reason, a quality plan should contain quality activities throughout the project.

Quality plans

- Defects can be injected in software at any stage during its evolution.
- These injection stages are primarily the requirements specification, the high-level design, the detailed design, and coding.
- For delivery of high-quality software, active removal of defects through the quality control activities is necessary.
- QC activities for defect removal are: requirements reviews, design reviews, code reviews, unit testing, integration testing, system testing, and acceptance testing (we do not include reviews of plan documents, although such reviews also help in improving quality of the software).
- With respect to quality control the terms verification and validation are often used. Together they are often called V&V activities.
- **Verification** is the process of determining whether or not the products of a given phase of software development fulfil the specifications established during the previous phase.
- **Validation** is the process of evaluating software at the end of the software development to ensure compliance with the software requirements.

Quality plans

- Clearly, for high reliability we need to perform both activities.
- Major V&V activities for software development are inspection and testing (both static and dynamic).
- The quality plan identifies the different V&V tasks for the different phases, how these tasks contribute to the project V&V goals, methods to be used for performing these V&V activities, the responsibilities and milestones for each of these activities, inputs and outputs for each V&V task, and criteria for evaluating the outputs.
- The quality plan specifies the quality control tasks that will be performed in the project.

6) Risk management

- A **risk** is a probabilistic event—it may or may not occur. It is defined as an exposure to the chance of injury or loss.
- **Risk management** is an attempt to minimize the chances of failure caused by unplanned events i.e. to minimize the impact of risks on cost, quality, and schedule of the undertaken project.
- Risk management can be considered as dealing with the possibility and actual occurrence of those events that are not "regular" or commonly expected, that is, they are probabilistic.
- It deals with events that are infrequent, somewhat out of the control of the project management, and which can have a major impact on the project.

Risk management

- **Risk assessment** is an activity that must be undertaken during project planning.
- This involves identifying the risks, analyzing them, and prioritizing them on the basis of the analysis.
- Due to the nature of a software project, uncertainties are highest near the beginning of the project (just as for cost estimation).
- So, although risk assessment should be done throughout the project, it is most needed in the starting phases of the project.
- Risk identification is the first step in risk assessment, which identifies all the different risks for a particular project.
- These risks are project-dependent and identifying them is an exercise in envisioning what can go wrong.
- Methods that can aid risk identification include checklists of possible risks, surveys, meetings and brainstorming, and reviews of plans, processes, and work products.

Risk management

- In **risk analysis**, the probability of occurrence of a risk has to be estimated, along with the loss that will occur if the risk does materialize.
- This is often done through discussion, using experience and understanding of the situation.
- If cost models are used for cost and schedule estimation, then the same models can be used to assess the cost and schedule risk. Once the probabilities of risks materializing and losses due to materialization of different risks have been analyzed, they can be prioritized.

Risk management

- Once a project manager has identified and prioritized the risks, the top risks can be easily identified.
- The question then becomes what to do about them.
- There can be strategies to follow:
  - ✓ risk avoidance: taking actions that will avoid the risk altogether
  - ✓ risk mitigation: to perform actions that will either reduce the probability of the risk materializing or reduce the loss due to the risk materializing.

Risk management

- Risk prioritization and consequent planning are based on the risk perception at the time the risk analysis is performed.
- As risks are probabilistic events and depend on external factors, the risk perception may also change with time.
- So, along with monitoring the progress of the planned risk mitigation steps, a project must periodically revisit the risk perception and modify the risk mitigation plans, if needed.
- **Risk monitoring** is the activity of monitoring the status of various risks and their control activities.

7) Project monitoring plans

- A project management plan is merely a document that can be used to guide the execution of a project.
- A good plan is useless unless properly executed.
- Execution is properly driven if it is monitored carefully and the actual performance is tracked against the plan.
- Monitoring requires measurements to be made to assess the situation of a project.
- For monitoring a project schedule, size, effort, and defects are the basic measurements that are needed.

Project monitoring plans

- Effort is the main resource consumed in a software project. And has to be tracked for evaluating whether the project is executing within budget.
- For effort data some type of timesheet system is needed where each person working on the project enters the amount of time spent on the project.
- For better monitoring, the effort spent on various tasks should be logged separately.
- Generally effort is recorded through some online system (e.g. weekly activity report system), which allows a person to record the amount of time spent on a particular activity.
- At any point, total effort on an activity can be aggregated.

Project monitoring plans

- Tracking of defects is critical for ensuring quality.
- A large software project may include thousands of defects that are found by different people at different stages.
- To keep track of the defects found and their status, defects must be logged and their closure tracked.
- Once each defect found is logged (and later closed), analysis can focus on how many defects have been found so far, what percentage of defects are still open and other issues.
- Defect tracking is considered one of the best practices for managing a project.

Project monitoring plans

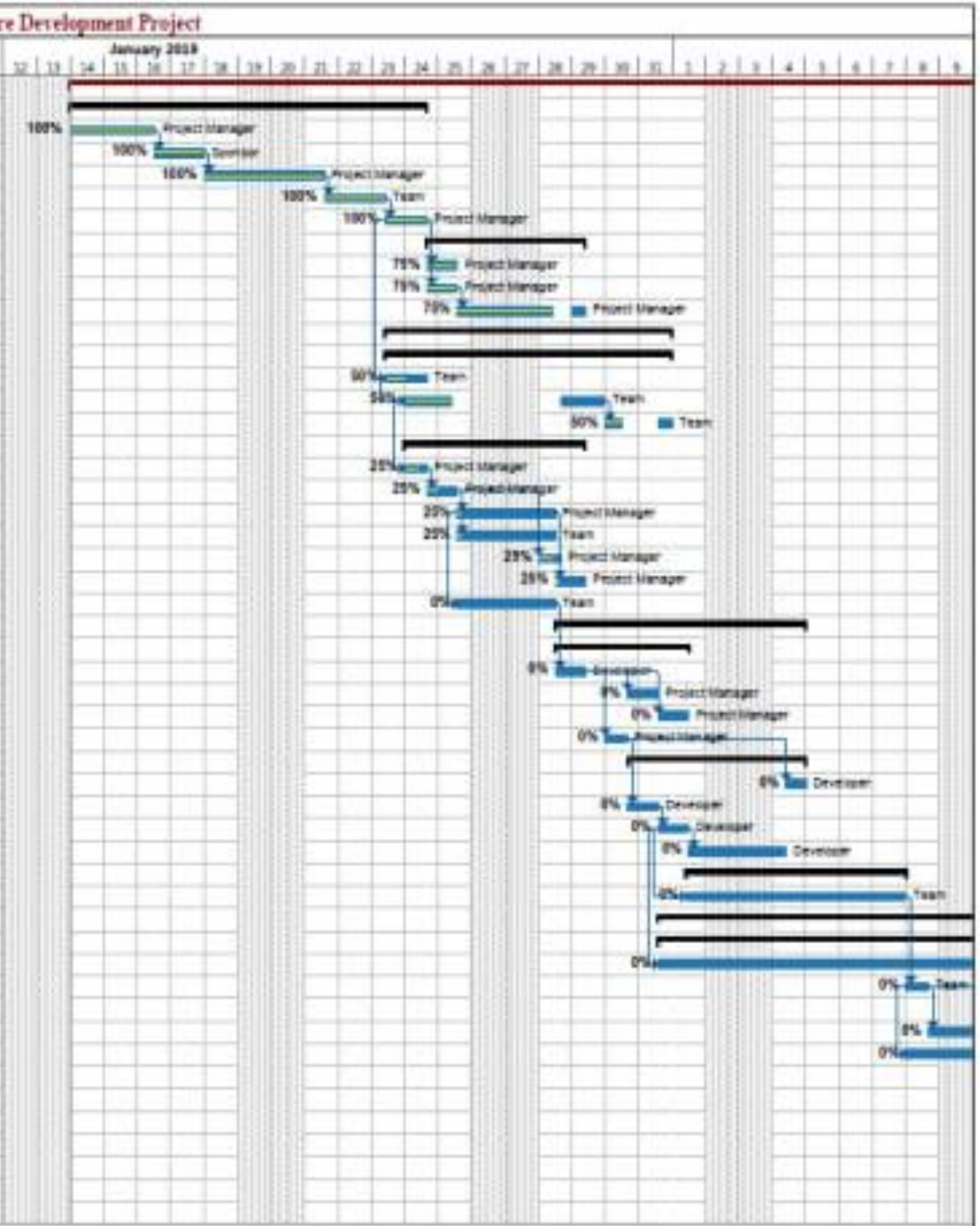
- Size is another fundamental metric because many data (for example, delivered defect density) are normalized with respect to size.
- The size of delivered software can be measured in terms of LOC or function points.
- At a more gross level, just the number of modules or number of features might suffice.

Project monitoring plans

- Different types of monitoring might be done for a project. The three main levels of monitoring are:
  - ✓ activity level
  - ✓ status reporting
  - ✓ milestone analysis
- Measurements taken on the project are employed for monitoring.
- A graphical method of capturing the basic progress of a project as compared to its plans is the cost-schedule-milestone graph. The graph gives the planned schedule and cost of different milestones, along with the actual cost and schedule of achieving the milestones achieved so far.

Software Development Project

ID	Task Name	Duration	Start	Finish	Resource Names	Cost
1	Software Development Project	162.75 days	Jan 14	Jul 15		$ 69,720
2	**Define Project / Opportunity**	**11 days**	**Jan 14**	**Jan 24**		**$ 3,200**
3	Understand / research the process of Arcade games	3 days	Jan 14	Jan 16	Project Manager	$ 1,200
4	Define its scope	2 days	Jan 16	Jan 17	Sponsor	$ 0
5	Identify Constraints	2 days	Jan 18	Jan 21	Project Manager	$ 800
6	Investigate existing players (if any)	2 days	Jan 21	Jan 23	Team	$ 400
7	Identify Risks	2 days	Jan 23	Jan 24	Project Manager	$ 800
8	**Gather Requirements**	**3 days**	**Jan 24**	**Jan 29**		**$ 1,200**
9	Research Systems and Games available in market	1 day	Jan 24	Jan 25	Project Manager	$ 400
10	Explore and select the mean of logistics	1 day	Jan 24	Jan 25	Project Manager	$ 400
11	Finalize the idea	1 day	Jan 25	Jan 29	Project Manager	$ 400
12	**Marketing**	**8.5 days**	**Jan 23**	**Jan 31**		**$ 1,200**
13	**Develop marketing strategy**	**8.5 days**	**Jan 23**	**Jan 31**		**$ 1,200**
14	Research the market of arcade games	2 days	Jan 23	Jan 24	Team	$ 400
15	Make Surveys of malls etc.	3 days	Jan 24	Jan 29	Team	$ 600
16	Identify locations	1 day	Jan 30	Jan 31	Team	$ 200
17	**Estimate Budget**	**4 days**	**Jan 24**	**Jan 29**		**$ 2,400**
18	Hardware cost	1 day	Jan 24	Jan 24	Project Manager	$ 400
19	Software's / Games cost	1 day	Jan 24	Jan 25	Project Manager	$ 400
20	Location Rent cost	1 day	Jan 25	Jan 28	Project Manager	$ 400
21	Logistics cost	1 day	Jan 25	Jan 28	Team	$ 200
22	Human Resources cost (if any)	1 day	Jan 28	Jan 28	Project Manager	$ 400
23	Installation and Commissioning cost	1 day	Jan 28	Jan 29	Project Manager	$ 400
24	Dismantling cost	1 day	Jan 25	Jan 28	Team	$ 200
25	**Design and Development**	**7 days**	**Jan 28**	**Feb 4**		**$ 2,800**
26	**Infrastructure Designing**	**5 days**	**Jan 28**	**Feb 1**		**$ 1,520**
27	Arcade Layout and Interior Design	1 day	Jan 28	Jan 29	Developer	$ 320
28	Make layout of Portable room / theater	1 day	Jan 30	Jan 31	Project Manager	$ 400
29	Make player station around 8' x 8' to 8'x 10'	1 day	Jan 31	Feb 1	Project Manager	$ 400
30	Select the appropriate material (less weighty)	1 day	Jan 30	Jan 30	Project Manager	$ 400
31	**Interior Designing**	**4 days**	**Jan 30**	**Feb 4**		**$ 1,280**
32	Select the appropeate Theme	1 day	Feb 4	Feb 4	Developer	$ 320
33	Select color combination	1 day	Jan 30	Jan 31	Developer	$ 320
34	Lighting (synchronizing with game play)	1 day	Jan 31	Feb 1	Developer	$ 320
35	Sound System (Synchronizing with game play)	1 day	Feb 1	Feb 4	Developer	$ 320
36	**Manufacturing**	**6 days**	**Feb 1**	**Feb 7**		**$ 1,200**
37	Manufacturing of rooms	6 days	Feb 1	Feb 7	Team	$ 1,200
38	**Search Hardware requirements specifications**	**33.75 days**	**Jan 31**	**Mar 11**		**$ 6,200**
39	**Rent / Purchase Equipment**	**33.75 days**	**Jan 31**	**Mar 11**		**$ 6,200**
40	Computer	15 days	Jan 31	Feb 18	Team	$ 3,000
41	Graphics Card	1 day	Feb 8	Feb 8	Team	$ 200
42	Screen/LED/Projector	1 day	Mar 8	Mar 11	Team	$ 200
43	Headsets	1 day	Feb 8	Feb 11	Project Manager	$ 400
44	Controllers	2 days	Feb 8	Feb 11	Project Manager	$ 800
45	Headphones	1 day	Feb 11	Feb 12	Project Manager	$ 400
46	Guns / Joy sticks	3 days	Feb 12	Feb 14	Team	$ 600
47	Any others	3 days	Feb 14	Feb 19	Team	$ 600
48	**Develop Software Interface**	**128 days**	**Feb 19**	**Jul 11**		**$ 11,120**
49	Communication protocols and interfaces	15 days	Feb 19	Mar 7	Project Manager	$ 6,000
50	Rent / Purchase Software / Games	2 days	Mar 7	Mar 8	Project Manager	$ 800
51	Gameplay setting	2 days	Mar 8	Mar 12	Project Manager	$ 800



Task ▬ Milestone • Summary ▬ Progress ▬ Project Summery ▬

COMPANY NAME

Project Plan: Software Development Project

Project Manager: ABC XYZ

Software Quality Assurance

Software Quality Assurance

- Software quality assurance (SQA) is an umbrella activity that is applied throughout the software process. SQA encompasses:
  - ✓ a quality management approach
  - ✓ effective software engineering technology (methods and tools)
  - ✓ formal technical reviews that are applied throughout the software process
  - ✓ a multi-tiered testing strategy
  - ✓ control of software documentation and the changes made to it,
  - ✓ a procedure to ensure compliance with software development standards (when applicable)
  - ✓ measurement and reporting mechanisms

Quality

- All engineered and manufactured parts exhibit variation.
- Variation control is the heart of quality control.
- Two kinds of quality may be encountered: *quality of design* and *quality of conformance*.

Quality

- ***Quality of design*** refers to the characteristics that designers specify for an item. The grade of materials, tolerances, and performance specifications all contribute to the quality of design.
- ***Quality of conformance*** is the degree to which the design specifications are followed during manufacturing.
- In software development, quality of design encompasses requirements, specifications, and the design of the system.
- Quality of conformance is an issue focused primarily on implementation. If the implementation follows the design and the resulting system meets its requirements and performance goals, conformance quality is high.

Quality control

- ***Quality control*** involves the series of inspections, reviews, and tests used throughout the software process to ensure each work product meets the requirements placed upon it.
- Quality control includes a feedback loop to the process that created the work product.
- The combination of measurement and feedback allows us to tune the process when the work products created fail to meet their specifications.
- This approach views quality control as part of the manufacturing process.

Quality assurance

- ***Quality assurance*** consists of the auditing and reporting functions of management.
- The goal of quality assurance is to provide management with the data necessary to be informed about product quality, thereby gaining insight and confidence that product quality is meeting its goals.
- If the data provided through quality assurance identify problems, it is management's responsibility to address the problems and apply the necessary resources to resolve quality issues.

Cost of quality

- Cost of quality includes all costs incurred in the pursuit of quality or in performing quality-related activities.
- Quality costs may be divided into costs associated with *prevention*, *appraisal*, and *failure*.

Cost of quality

- Prevention costs include:
  - ✓quality planning
  - ✓formal technical reviews
  - ✓test equipment
  - ✓training

Cost of quality

- Appraisal costs include activities to gain insight into product condition the “first time through” each process.
- Examples of *appraisal costs* include
  - ✓in-process and interprocess inspection
  - ✓equipment calibration and maintenance
  - ✓testing

Cost of quality

- Failure costs are those that would disappear if no defects appeared before shipping a product to customers.
- Failure costs may be subdivided into internal failure costs and external failure costs.
- Internal failure costs are incurred when we detect a defect in our product prior to shipment.
- Internal failure costs include
  - ✓ rework
  - ✓ repair
  - ✓ failure mode analysis

Cost of quality

- External failure costs are associated with defects found after the product has been shipped to the customer.
- Examples of external failure costs are:
  - ✓ complaint resolution
  - ✓ product return and replacement
  - ✓ help line support
  - ✓ warranty work

Software Quality

Software Quality is defined as:

"*Conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software.*"

Software Quality Assurance

The activities performed by an independent SQA group are:

a) Prepares an SQA plan for a project. The plan identifies:

- ✓ evaluations to be performed
- ✓ audits and reviews to be performed
- ✓ standards that are applicable to the project
- ✓ procedures for error reporting and tracking
- ✓ documents to be produced by the SQA group
- ✓ amount of feedback provided to the software project team

Software Quality Assurance

- b) Participates in the development of the project’s software process description
- c) Reviews software engineering activities to verify compliance with the defined software process
- d) Audits designated software work products to verify compliance with those defined as part of the software process

Software Quality Assurance

- e) Ensures that deviations in software work and work products are documented and handled according to a documented procedure
- f) Records any noncompliance and reports to senior management

Formal Technical Reviews

- A formal technical review is a software quality assurance activity performed by software engineers (and others).
- FTR objectives:
  - ✓ to uncover errors in function, logic, or implementation for any representation of the software;
  - ✓ to verify that the software under review meets its requirements;
  - ✓ to ensure that the software has been represented according to predefined standards;
  - ✓ to achieve software that is developed in a uniform manner;
  - ✓ to make projects more manageable.

Formal Technical Reviews

- FTR meeting constraints:
  - ✓ Between three and five people (typically) should be involved in the review.
  - ✓ Advance preparation should occur but should require no more than two hours of work for each person.
  - ✓ The duration of the review meeting should be less than two hours.

Formal Technical Reviews

- The focus of the FTR is on a work product.
- The individual who has developed the work product—the producer—informs the project leader that the work product is complete and that a review is required.
- The project leader contacts a review leader, who evaluates the product for readiness, generates copies of product materials, and distributes them to two or three reviewers for advance preparation.
- Each reviewer is expected to spend between one and two hours reviewing the product, making notes, and otherwise becoming familiar with the work.

Formal Technical Reviews

- Concurrently, the review leader also reviews the product and establishes an agenda for the review meeting, which is typically scheduled for the next day.
- The review meeting is attended by the review leader, all reviewers, and the producer.
- One of the reviewers takes on the role of the recorder; that is, the individual who records (in writing) all important issues raised during the review.
- The FTR begins with an introduction of the agenda and a brief introduction by the producer, then proceeds to "walk through" the work product, explaining the material, while reviewers raise issues based on their advance preparation.
- When valid problems or errors are discovered, the recorder notes each.

Formal Technical Reviews

- At the end of the review, all attendees of the FTR must decide whether to
  - ✓ ***accept*** the product without further modification,
  - ✓ ***reject*** the product due to severe errors (once corrected, another review must be performed), or
  - ✓ ***accept*** the product ***provisionally*** (minor errors have been encountered and must be corrected, but no additional review will be required).

Formal Technical Reviews

- A formal technical review summary report is completed. A review summary report answers three questions:
  - ✓ What was reviewed?
  - ✓ Who reviewed it?
  - ✓ What were the findings and conclusions?
- The review summary report is a single page form (with possible attachments).
- It becomes part of the project historical record and may be distributed to the project leader and other interested parties.

Formal Technical Reviews

- The review issues list serves two purposes:
  - ✓ to identify problem areas within the product
  - ✓ to serve as an action item checklist that guides the producer as corrections are made.
- An issues list is normally attached to the summary report.

Formal Technical Reviews

- Possible causes of errors:
  - ✓ incomplete or erroneous specifications (IES)
  - ✓ misinterpretation of customer communication (MCC)
  - ✓ intentional deviation from specifications (IDS)
  - ✓ violation of programming standards (VPS)
  - ✓ error in data representation (EDR)
  - ✓ inconsistent component interface (ICI)
  - ✓ error in design logic (EDL)
  - ✓ incomplete or erroneous testing (IET)
  - ✓ inaccurate or incomplete documentation (IID)
  - ✓ error in programming language translation of design (PLT)
  - ✓ ambiguous or inconsistent human/computer interface (HCI)
  - ✓ miscellaneous (MIS)

Software Reliability

- Software reliability is defined in statistical terms as *"the probability of failure-free operation of a computer program in a specified environment for a specified time"*.

Software Reliability

- Failure is non conformance to software requirements.
- Failures can be only annoying or catastrophic.
- One failure can be corrected within seconds while another requires weeks or even months to correct.
- Complicating the issue even further, the ***correction of one failure may in fact result in the introduction of other errors that ultimately result in other failures***.

Software Reliability

- A simple measure of reliability is **meantime-between-failure (MTBF)**, where:

MTBF = MTTF + MTTR

MTTF => mean-time-to-failure

MTTR => mean-time-to-repair

- **Software availability** is the probability that a program is operating according to requirements at a given point in time and is defined as:

Availability = [MTTF/(MTTF + MTTR)] X 100%

- The MTBF reliability measure is equally sensitive to MTTF and MTTR.
- The availability measure is somewhat more sensitive to MTTR, an indirect measure of the maintainability of software.

SQA Plan

- SQA plan serves as a template for SQA activities that are instituted for each software project
- IEEE standard for SQA plan:
  - ✓ Initial sections
  - ✓ Management section
  - ✓ Documentation section
  - ✓ Standards, practices, and conventions section
  - ✓ Reviews and audits section
  - ✓ Test section
  - ✓ Problem reporting and corrective action
  - ✓ Others

SQA Plan- contents

- **Initial sections**
  - ✓ purpose and scope of the document and indicate those software process activities that are covered by quality assurance

- **Management section**
  - ✓ SQA’s place in the organizational structure, SQA tasks and activities and their placement throughout the software process, and the organizational roles and responsibilities relative to product quality.

SQA Plan- contents

- **Documentation section**
  - ✓ describes the work products produced as part of the software process like:
    - project documents (e.g., project plan)
    - models (e.g., ERDs, class hierarchies)
    - technical documents (e.g., specifications, test plans)
    - user documents (e.g., help files)
  - ✓ defines the minimum set of work products that are acceptable to achieve high quality

SQA Plan- contents

- **Standards, practices, and conventions section**
  - ✓ lists all applicable standards and practices that are applied during the software process (e.g., document standards, coding standards, and review guidelines).
  - ✓ All project, process, and product metrics that are to be collected as part of software engineering work are listed.

- **Reviews and audits section**
  - ✓ identifies the reviews and audits to be conducted by the software engineering team, the SQA group, and the customer.
  - ✓ provides an overview of the approach for each review and audit.

SQA Plan- contents

- **Test section**
  - ✓ references the Software Test Plan and Procedure.
  - ✓ defines test record-keeping requirements.

- **Problem reporting and corrective action**
  - ✓ defines procedures for reporting, tracking, and resolving errors and defects
  - ✓ identifies the organizational responsibilities for these activities.

SQA Plan- contents

- **Others**
  - ✓ tools and methods that support SQA activities and tasks
  - ✓ references software configuration management procedures for controlling change
  - ✓ defines a contract management approach
  - ✓ establishes methods for assembling, safeguarding, and maintaining all records
  - ✓ identifies training required to meet the needs of the plan
  - ✓ defines methods for identifying, assessing, monitoring, and controlling risk.

Software Configuration Management

Software configuration management (SCM)

- Software configuration management (SCM) is an umbrella activity that is applied throughout the software process.
- It is a set of tracking and control activities that begin when a software engineering project begins and terminate only when the software is taken out of operation.
- SCM activities are developed to:
  - ✓ identify change
  - ✓ control change
  - ✓ ensure that change is being properly implemented
  - ✓ report changes to others who may have an interest.

SCM

- Output of the software process is information that may be divided into three broad categories:
  - ✓ computer programs (both source level and executable forms)
  - ✓ documents that describe the computer programs (targeted at both technical practitioners and users)
  - ✓ data (contained within the program or external to it).

Software Configuration Items (SCIs)

- The items that comprise all information produced as part of the software process are collectively called a software configuration.
- As the software process progresses, the number of **software configuration items (SCIs)** grows rapidly.

Changes

- There are four fundamental sources of change:
  - ✓ New business or market conditions dictate changes in product requirements or business rules.
  - ✓ New customer needs demand modification of data produced by information systems, functionality delivered by products, or services delivered by a computer-based system.
  - ✓ Reorganization or business growth/downsizing causes changes in project priorities or software engineering team structure.
  - ✓ Budgetary or scheduling constraints cause a redefinition of the system or product.

Baselines

- A baseline can be defined as: “*A specification or product that has been formally reviewed and agreed upon, that thereafter serves as the basis for further development, and that can be changed only through formal change control procedures*”.
- Before a software configuration item becomes a baseline, change may be made quickly and informally.
- However, once a baseline is established, changes can be made, but a specific, formal procedure must be applied to evaluate and verify each change.
- After SCIs are reviewed and approved, they are placed in a project database (also called a project library or software repository).

Software Configuration Items

- An SCI is a document, a entire suite of test cases, or a named program component.
- SCIs are organized to form configuration objects that may be catalogued in the project database with a single name.
- A configuration object has a name, attributes, and is "connected" to other objects by relationships.
- The configuration objects, Design Specification, data model, component N, source code and Test Specification are each defined separately.
- Each of the objects is related to the others. The relationship may be compositional.
- If a change were made to the source code object, the interrelationships enable a software engineer to determine what other objects (and SCIs) might be affected.

SCM Process

- ➢ The change management process has the following steps.
  - ✓ Log the changes
  - ✓ Perform impact analysis on the work products
  - ✓ Estimate impact on effort and schedule
  - ✓ Review impact with concerned stakeholders
  - ✓ Rework work products

SCM Process

- A change is initiated by a change request.
- A change request log is maintained to keep track of the change requests.
- Each entry in the log contains a change request number, a brief description of the change, the effect of the change, the status of the change request, and key dates.
- SCM is responsible for the identification of individual SCIs and various versions of the software, the auditing of the software configuration to ensure that it has been properly developed, and the reporting of all changes applied to the configuration.

SCM Process

- To control and manage software configuration items, each must be separately named and then organized using an object-oriented approach.
- Object types: **basic objects** and **aggregate objects**
- A *basic object* might be a section of a requirements specification, a source listing for a component, or a suite of test cases that are used to exercise the code.
- An *aggregate object* is a collection of basic objects and other aggregate objects.

SCM Process

- Each object has a set of distinct features that identify it uniquely: *a name, a description, a list of resources, and a "realization."* The object name is a character string that identifies the object unambiguously. The object description is a list of data items that identify:
  - ✓ the SCI type (e.g., document, program, data) represented by the object
  - ✓ a project identifier
  - ✓ change and/or version information
- Resources are "entities that are provided, processed, referenced or otherwise required by the object." For example, data types, specific functions, or even variable names may be considered to be object resources.
- The realization is a pointer to the "unit of text" for a basic object and null for an aggregate object.

Version Control

- Version control combines procedures and tools to manage different versions of configuration objects that are created during the software process.
- Each version of the software is a collection of SCIs (source code, documents, data), and each version may be composed of different variants.
- An entity is composed of a collection of objects at the same revision level.
- A variant is a different collection of objects at the same revision level and therefore coexists in parallel with other variants.
- A new version is defined when major changes are made to one or more objects.

Change Control Process

- A change request is submitted and evaluated to assess technical merit, potential side effects, overall impact on other configuration objects and system functions, and the projected cost of the change.
- Results of the evaluation are presented as a change report, which is used by a change control authority (CCA)—a person or group who makes a final decision on the status and priority of the change.
- An engineering change order (ECO) is generated for each approved change.
- ECO describes the change to be made, the constraints that must be respected, and the criteria for review and audit.

Change Control Process

- The object to be changed is "checked out" of the project database, the change is made, and appropriate SQA activities are applied.
- The object is then "checked in" to the database and appropriate version control mechanisms are used to create the next version of the software.
- The "check-in" and "check-out" process implements two important elements of change control—access control and synchronization control.
- **Access control** governs which software engineers have the authority to access and modify a particular configuration object.
- **Synchronization control** helps to ensure that parallel changes, performed by two different people, don't overwrite one another.
- Other copies can be checked-out, but other updates cannot be made.

Change Control Process

- A copy of the baselined object, called the extracted version, is modified by the software engineer.
- After appropriate SQA and testing, the modified version of the object is checked in and the new baseline object is unlocked.
- Prior to an SCI becoming a baseline, only informal change control need be applied.
- The developer of the configuration object (SCI) in question may make whatever changes are justified by project and technical requirements (as long as changes do not affect broader system requirements that lie outside the developer's scope of work).
- Once the object has undergone formal technical review and has been approved, a baseline is created.
- Once an SCI becomes a baseline, project level change control is implemented.
- Now, to make a change, the developer must gain approval from the project manager (if the change is "local") or from the CCA if the change affects other SCIs.
- Assessment of each change is conducted and all changes are tracked and reviewed.
- When the software product is released to customers, formal change control is instituted.

Configuration Audit

How to ensure that the change has been properly implemented:

- formal technical reviews
- software configuration audit.

Configuration Audit

- Formal technical review focuses on the technical correctness of the configuration object that has been modified.
- The reviewers assess the SCI to determine consistency with other SCIs, omissions, or potential side effects.
- A formal technical review should be conducted for all but the most trivial changes

Configuration Audit

- A software configuration audit complements the formal technical review by assessing a configuration object for characteristics that are generally not considered during review.
- When SCM is a formal activity, the SCM audit is conducted separately by the quality assurance group.

Configuration Audit

- Typical questions asked in an SCM audit:
  - Has the change specified in the ECO been made? Have any additional modifications been incorporated?
  - Has a formal technical review been conducted to assess technical correctness?
  - Has the software process been followed and have software engineering standards been properly applied?
  - Has the change been "highlighted" in the SCI? Have the change date and change author been specified? Do the attributes of the configuration object reflect the change?
  - Have SCM procedures for noting the change, recording it, and reporting it been followed?
  - Have all related SCIs been properly updated?

Status Reporting

- Configuration status reporting (sometimes called status accounting) is an SCM task that answers the following questions:
  - ✓What happened?
  - ✓Who did it?
  - ✓When did it happen?
  - ✓What else will be affected?

Status Reporting

- Each time an SCI is assigned new or updated identification, a CSR entry is made.
- Each time a change is approved by the CCA (i.e., an ECO is issued), a CSR entry is made.
- Each time a configuration audit is conducted, the results are reported as part of the CSR task.
- Output from CSR may be placed in an on-line database so that software developers or maintainers can access change information by keyword category.
- In addition, a CSR report is generated on a regular basis and is intended to keep management and practitioners appraised of important changes.

Thank You...