**Unit 4 - Project Management and Estimation**

Overview

Project management involves the planning, monitoring, and control of people, process, and events that occur during software development.

Everyone manages, but the scope of each person's management activities varies according his or her role in the project.

Software needs to be managed because it is a complex undertaking with a long duration time. Managers must focus on the fours P's to be successful (people, product, process, and project). A project plan is a document that deﬁnes the four P's in such a way as to ensure a cost effective, high quality software product.

The only way to be sure that a project plan worked correctly is by observing that a high quality product was delivered on time and under budget.

Management Spectrum

People (recruiting, selection, performance management, training, compensation, career development, organization, work design, team/culture development)

Product (product objectives, scope, alternative solutions, constraint tradeoffs)

Process (framework activities populated with tasks, milestones, work products, and QA points) Project (planning, monitoring, controlling)

People

Players (senior managers, technical managers, practitioners, customers, end-users) Team leadership model (motivation, organization, skills)

Characteristics of effective project managers (problem solving, managerial identity, achievement, inﬂuence and team building)

Software Team Organization

Democratic decentralized (rotating task coordinators and group consensus)

Controlled decentralized (permanent leader, group problem solving, subgroup implementation of solutions)

Controlled centralized (top level problem solving and internal coordination managed by team leader)

Factors Affecting Team Organization

Diﬃculty of problem to be solved Size of resulting program

Team lifetime

Degree to which problem can be modularized

Required quality and reliability of the system to be built

Rigidity of the delivery date

Degree of communication required for the project Coordination and Communication Issues

Formal, impersonal approaches (e.g. documents, milestones, memos) Formal interpersonal approaches (e.g. review meetings, inspections)

Informal interpersonal approaches (e.g. information meetings, problem solving) Electronic communication (e.g. e-mail, bulletin boards, video conferencing)

Interpersonal network (e.g. informal discussion with people other than project team members)

The Product

Software scope (context, information objectives, function, performance)

Problem decomposition (partitioning or problem elaboration - focus is on functionality to be delivered and the process used to deliver it)

The Process

Process model chosen must be appropriate for the: customers and developers, characteristics of the product, and project development environment

Project planning begins with melding the product and the process

Each function to be engineered must pass though the set of framework activities deﬁned for a software organization

Work tasks may vary but the common process framework (CPF) is invariant (project size does not change the CPF)

The job of the software engineer is to estimate the resources required to move each function though the framework activities to produce each work product

Project decomposition begins when the project manager tries to determine how to accomplish each CPF activity

The Project

Start on the right foot Maintain momentum Track progress

Make smart decisions

Conduct a postmortem analysis

W5HH Principle

Why is the system being developed? What will be done by When?

Who is responsible for a function? Where are they organizationally located?

How will the job be done technically and managerially? How much of each resource is needed?

Critical Practices

Formal risk management

Empirical cost and schedule estimation Metric-based project management Earned value tracking

Defect tracking against quality targets People-aware program management

**Estimation for Software Projects (Chapter 23)**

***Overall goal*** *- to establish a pragmatic strategy for controlling, tracking, and monitoring a complex technical project.*

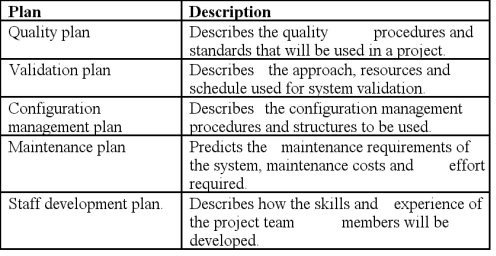
*Most time-consuming project management activity*

*Continuous activity from initial concept through to system delivery.*

*Plans must be regularly revised as new information becomes available*

*Various different types of plan may be developed to support the main software project plan that is concerned with schedule and budget*

**Types of project plan**



**Steps in Process**

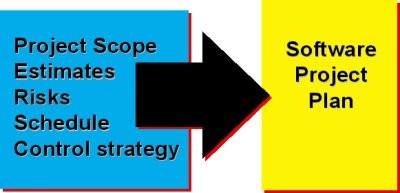
***Scoping*** *— understand the problem and the work that must be done*

***Estimation*** *— how much effort? how much time?*

***Risk*** *— what can go wrong? how can we avoid it? what can we do about it?*

***Schedule*** *— how do we allocate resources along the timeline? what are the milestones?*

***Control strategy*** *— how do we control quality? how do we control change?*



## Understanding Scope

*understand the customers needs understand the business context understand the project boundaries*

*understand the customer’s motivation understand the likely paths for change*

*understand that ... nothing is guaranteed!*

## Cost Estimation

*project scope must be explicitly deﬁned*

*task and/or functional decomposition is necessary historical measures (metrics) are very helpful*

*at least two different techniques should be used remember that uncertainty is inherent*

## Estimation Techniques

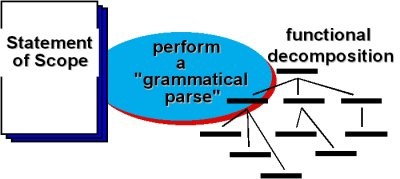
**past** (similar) project experience

**conventional** estimation techniques

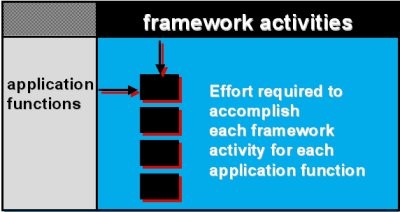
task breakdown and effort estimates size (e.g., FP) estimates

### empirical models automated tools

**Functional Decomposition**



**Creating a Task Matrix (Obtained from “process framework”)**



**Conventional Methods: LOC/FP Approach**

*compute LOC/FP using estimates of information domain values use historical effort for the project*

## Example: LOC Approach e.g.

Average productivity for systems of this type = 620 LOC/pm.

Labor rate =$8000 per month, the cost per line of code is approximately $13.

Based on the LOC estimate and the historical productivity data, the total estimated project cost is $431,000 and the estimated effort is 54 person-months.

**e.g.**



**Example: FP Approach e.g.**

The estimated number of FP is derived:

FPestimated = count-total 3 [0.65 + 0.01 3 S (Fi)]

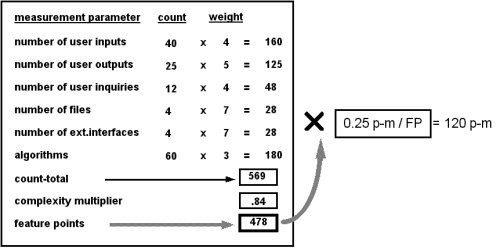
FPestimated = 375

organizational average productivity = 6.5 FP/pm.

labor rate = $8000 per month, the cost per FP is approximately $1230.

Based on the FP estimate and the historical productivity data, the total estimated project cost is $461,000 and the estimated effort is 58 person-months.

## e.g.

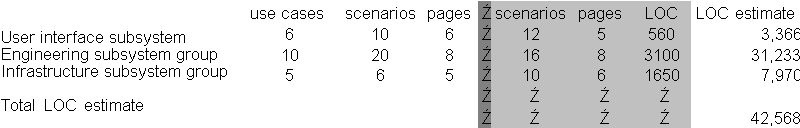


**Tool-Based Estimation**

*project characteristics calibration factors*

*LOC/FP data*

**Estimation with Use-Cases**



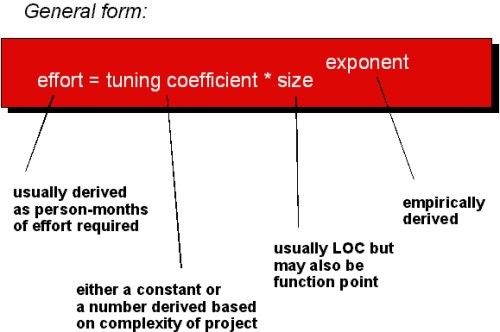
Using 620 LOC/pm as the average productivity for systems of this type Labor rate of $8000 per month

the cost per line of code is approximately $13.

Based on the use-case estimate and the historical productivity data, the total estimated project cost is $552,000

Estimated effort is 68 person-months.

## Empirical Estimation Models



**COCOMO II**

* **COCOMO** is a model designed by [Barry Boehm](http://en.wikipedia.org/wiki/Barry_Boehm) to give an estimate of the number of programmer-months it will take to develop a software product.
* This "**COnstructive COst MOdel**" is based on a study of about sixty projects at [TRW](http://en.wikipedia.org/wiki/TRW), a Californian automotive and IT company, acquired by [Northrop Grumman](http://en.wikipedia.org/wiki/Northrop_Grumman) in late 2002. The programmes examined ranged in size from 2000 to 100,000 lines of code, and programming languages used ranged from assembly to PL/I.
* Cocomo consists of a hierarchy of three increasingly detailed and accurate forms.

[Basic COCOMO](http://en.wikipedia.org/wiki/Basic_COCOMO) - is a static single-valued model that computes software development effort (and cost) as a function of program size expressed in estimated lines of code.

[Intermediate COCOMO](http://en.wikipedia.org/wiki/Intermediate_COCOMO) - computes software development effort as function of program size and a set of "cost drivers" that include subjective assessment of product, hardware, personnel and project attributes.

[Detailed COCOMO](http://en.wikipedia.org/w/index.php?title=Detailed_COCOMO&amp;action=edit) - incorporates all characteristics of the intermediate version with an assessment of the cost driver's impact on each step (analysis, design, etc.) of the software engineering process.

**Important observation:** personnel motivation overwhelms all other parameters.

* + suggests that leadership and teamsmanship are the most important skills of all, but this point was largely ignored.

**Estimation for OO Projects-I**

* + Develop estimates using effort decomposition, FP analysis, and any other method that is applicable for conventional applications.

1. Using object-oriented analysis modeling (Chapter 8), develop use-cases and determine a count.
2. From the analysis model, determine the number of key classes (called analysis classes in Chapter 8).
3. Categorize the type of interface for the application and develop a multiplier for support classes:

|  |  |
| --- | --- |
| **Interface type** | **Multiplier** |
| No GUI | 2.0 |
| Text-based user interface | 2.25 |
| GUI | 2.5 |
| Complex GUI | 3.0 |

1. Multiply the number of key classes (step 3) by the multiplier to obtain an estimate for the number of support classes.
2. Multiply the total number of classes (key + support) by the average number of work-units per class. Lorenz and Kidd suggest 15 to 20 person-days per class.
3. Cross check the class-based estimate by multiplying the average number of work-units per use-case

**Estimation for Agile Projects**

* + Each user scenario (a mini-use-case) is considered separately for estimation purposes.
  + The scenario is decomposed into the set of software engineering tasks that will be required to develop it.
  + Each task is estimated separately. Note: estimation can be based on historical data, an empirical model, or “experience.”
    - Alternatively, the ‘volume’ of the scenario can be estimated in LOC, FP or some other volume-oriented measure (e.g., use-case count).
  + Estimates for each task are summed to create an estimate for the scenario.
    - Alternatively, the volume estimate for the scenario is translated into effort using historical data.
  + The effort estimates for all scenarios that are to be implemented for a given software increment are summed to develop the effort estimate for the increment.

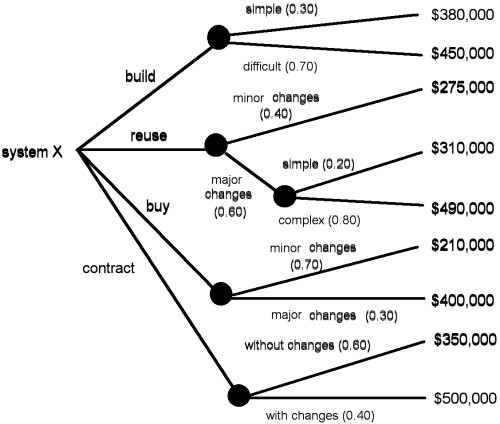
## Estimation Guidelines

*estimate using at least two techniques*

*get estimates from independent sources avoid over-optimism, assume diﬃculties you've arrived at an estimate, sleep on it*

*adjust for the people who'll be doing the job*

## The Make-Buy Decision



**Computing Expected Cost**

**expected cost** = SUM[ (path probability)i x (estimated path cost)i ] For example, the expected cost to build is:

**expected costbuild** = 0.30($380K)+0.70($450K) = $429 K similarly,

**expected costreuse** = $382K

**expected costbuy** = $267K

**expected costcontract** = $410K

# Software Project Scheduling (Chapter 24)

### Basic Concepts

**Causes of Software Lateness:**

An unrealistic deadline established by someone outside the software development group and forced on managers and practitioners within the group.

Changing customer requirements that are not reﬂected in schedule changes.

An honest underestimate of the amount of effort and/or the number of resources that will be required to do the job.

Predictable and/or unpredictable risks that were not considered when the project commenced. Technical diﬃculties that could not have been foreseen in advance.

Human diﬃculties that could not have been foreseen in advance. Miscommunication among project staff that results in delays.

A failure by project management to recognize that the project is falling behind schedule and a lack of action to correct the problem.

* ***Software project scheduling*** *-* activity that distributes estimated effort across the planned project duration by allocating the effort to speciﬁc software engineering tasks.
* Schedule evolves over time.
* Early stages of planning - ***macroscopic schedule*** - identiﬁes all major software engineering activities and the product functions to which they are applied.
* Project under way - ***detailed schedule*** *-* each entry on the macroscopic schedule is reﬁned*.* Here, s

§ speciﬁc software tasks are identiﬁed and scheduled.

### Two Points of view for Scheduling

1. End-date for release of a computer-based system has already (and irrevocably) been established.
2. Rough chronological bounds have been discussed but that the end-date is set by the software engineering

organization. Effort is distributed to make best use of resources and an end-date is deﬁned after careful analysis of the software.

### Principles guiding software project scheduling:

* **Compartmentalization.** The project must be compartmentalized into a number of manageable activities and tasks.
* **Interdependency.** The interdependency of each compartmentalized activity or task must be determined.

Some tasks must occur in sequence while others can occur in parallel.

### Time allocation.

§ Each task to be scheduled allocated some number of work units

§ Each task assigned a start date and a completion date that are a function of the interdependencies and whether work will be conducted on a full-time or part-time basis.

### Effort validation.

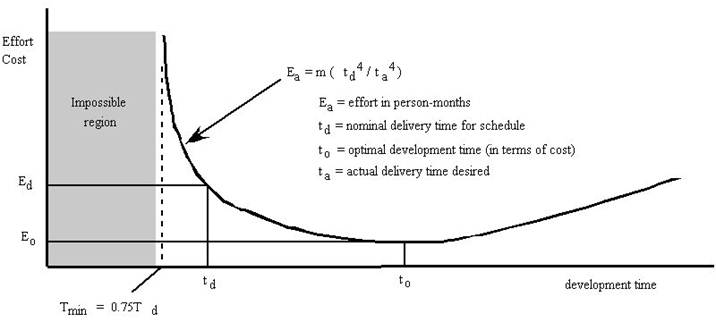
§ Every project has a deﬁned number of staff members.

§ As time allocation occurs, the project manager ensures that no more than the allocated number of people has been scheduled at any given time.

* **Deﬁned responsibilities.** Every task that is scheduled should be assigned to a speciﬁc team member.
* **Deﬁned outcomes.** Every task that is scheduled should have a deﬁned outcome. For software projects, the outcome is normally a work product or a part of a work product.
* **Deﬁned milestones.** Every task or group of tasks should be associated with a project milestone. A milestone

is accomplished when one or more work products has been reviewed for quality and has been approved.

**Effort and Delivery Time**



* ***40–20–40 rule*** - Recommended distribution of effort across the deﬁnition and development phases.
* Forty percent of all effort is allocated to front-end analysis and design.
* Forty percent is applied to back-end testing.
* Twenty percent is applied to coding.

### Deﬁning a Task Set for the Software Project

* ***Task set*** *-* collection of software engineering work tasks, milestones, and deliverables that must be accomplished to complete a particular project.
* Task sets are designed to accommodate different types of projects and different degrees of rigor.

o determine type of project

o assess the degree of rigor required

o identify adaptation criteria

select appropriate software engineering tasks

### Most software organizations encounter the following projects:

1. *Concept development projects* - initiated to explore some new business concept or application of some new technology.
2. *New application development projects* - undertaken as a consequence of a speciﬁc customer request.
3. *Application enhancement projects* - occur when existing software undergoes major modiﬁcations to function, performance, or interfaces that are observable by the end-user.
4. *Application maintenance projects* - correct, adapt, or extend existing software in ways that may not be immediately obvious to the end-user.
5. *Reengineering projects* - undertaken with the intent of rebuilding an existing (legacy) system in whole or in part.

* ***Degree of rigor*** is a function of many project characteristics.

### Four different degrees of rigor can be deﬁned:

1. **Casual.** All process framework activities are applied, but only a minimum task set is required. In general, umbrella tasks will be minimized and documentation requirements will be reduced. All basic principles of software engineering are still applicable.
2. **Structured.** The process framework will be applied for this project. Framework activities and related tasks appropriate to the project type will be applied and umbrella activities necessary to ensure high quality will be applied. SQA, SCM, documentation, and measurement tasks will be conducted in a streamlined manner.
3. **Strict.** The full process will be applied for this project with a degree of discipline that will ensure high quality. All umbrella activities will be applied and robust work products will be produced.
4. **Quick reaction.** The process framework will be applied for this project, but because of an emergency situation only those tasks essential to maintaining good quality will be applied. "Back-ﬁlling" (i.e., developing a complete set of documentation, conducting additional reviews) will be accomplished after the application/product is

delivered to the customer.

* Project manager must develop a systematic approach for selecting the degree of rigor that is appropriate for a particular project.
* To accomplish this, **project adaptation criteria** are deﬁned and a task set selector value is computed
* ***Adaptation criteria*** are used to determine the recommended degree of rigor with which the software process should be applied on a project.

**Eleven adaptation criteria** are deﬁned for software projects:

1. Size of the project
2. Number of potential users
3. Mission criticality
4. Application longevity
5. Stability of requirements
6. Ease of customer/developer communication
7. Maturity of applicable technology
8. Performance constraints
9. Embedded and nonembedded characteristics
10. Project staff
11. Reengineering factors

* Each adaptation criterion is assigned a grade that ranges between 1 and 5

**1** represents a project in which a small subset of process tasks are required and overall methodological and documentation requirements are minimal

**5** represents a project in which a complete set of process tasks should be applied and overall methodological and documentation requirements are substantial

### Task Selector Table

* **To select the appropriate task set for a project, the following steps should be conducted:**
  1. Review each of the adaptation criteria and assign the appropriate grades (1 to 5) based on the characteristics of the project.
     + Enter grade into table.
  2. Review the weighting factors assigned to each of the criteria.
     + Value of a weighting factor ranges from 0.8 to 1.2 and provides an indication of the relative importance of a particular adaptation criterion to the types of software developed within the local environment.
  3. Multiply the grade entered in table by the ***weighting factor*** and by the ***entry point multiplier*** for the type of project to be undertaken.
     + Entry point multiplier takes on a value of 0 or 1 and indicates the relevance of the adaptation criterion to

the project type.

* + - Result of the product

### grade x weighting factor x entry point multiplier

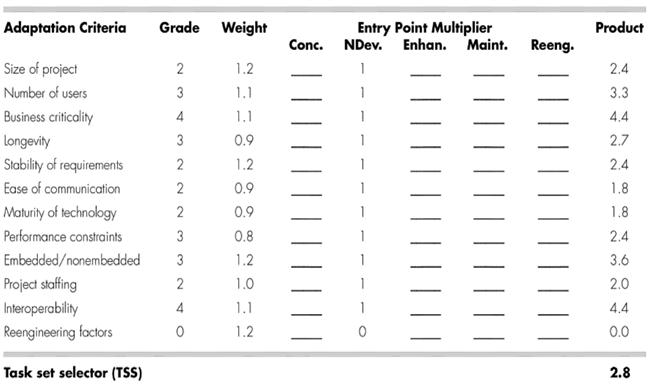
* + - Placed in the Product column of table for each adaptation criteria individually.
  1. Compute the average of all entries in the Product column and place the result in the space marked ***task set selector*** (TSS).
* This value will be used to help select the task set that is most appropriate for the project.

### Interpreting the TSS Value and Selecting the Task Set

* Once the task set selector is computed, the following guidelines can be used to select the appropriate task set for a project:

|  |  |
| --- | --- |
| ***Task set selector***  ***value*** | ***Degree of rigor*** |
| ***TSS < 1.2*** | casual |
| ***1.0 < TSS < 3.0*** | structured |
| ***TSS > 2.4*** | strict |

* Overlap in TSS values from one recommended task set to another is intended to illustrate that sharp boundaries are impossible to deﬁne when making task set selections.
* The task set selector value, past experience, and common sense must all be factored into the choice of the task set for a project.



* Table illustrates how TSS might be computed for a hypothetical project.
* Project manager selects the grades shown in the Grade column.
* Project type is *new application development.*
* Therefore, entry point multipliers are selected from the NDev column. T

o he entry in the Product column is computed using

### Grade x Weight x NewDev entry point multiplier

* Value of TSS (computed as the average of all entries in the product column) is 2.8.
* The manager has the option of using either the structured or the strict task set.

### Selecting Software Engineering Tasks

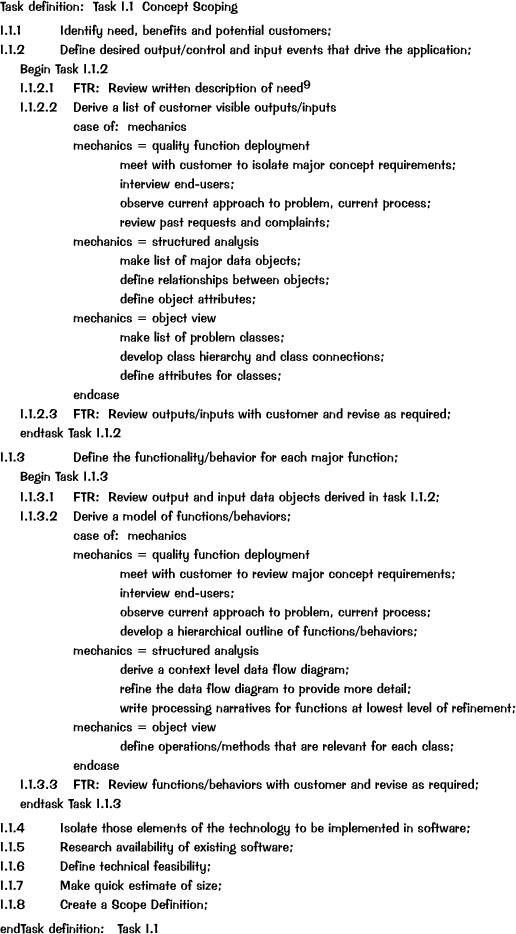
* **Development projects are approached by applying the following major tasks:**

1. **Concept scoping** determines the overall scope of the project.
2. **Preliminary concept planning** establishes the organization's ability to undertake the work implied by the project scope.
3. **Technology risk assessment** evaluates the risk associated with the technology to be implemented as part of project scope.
4. **Proof of concept** demonstrates the viability of a new technology in the software context.
5. **Concept implementation** implements the concept representation in a manner that can be reviewed by a customer and is used for "marketing" purposes when a concept must be sold to other customers or management.
6. **Customer reaction to the concept** solicits feedback on a new technology concept and targets speciﬁc customer applications.

* Development framework activities are iterative in nature.
* Actual development project might approach these activities in a number of planned increments, each designed to produce a deliverable that can be evaluated by the customer.
* Model Selection:
* Linear process model
* Evolutionary model

### Reﬁnement of Major Tasks

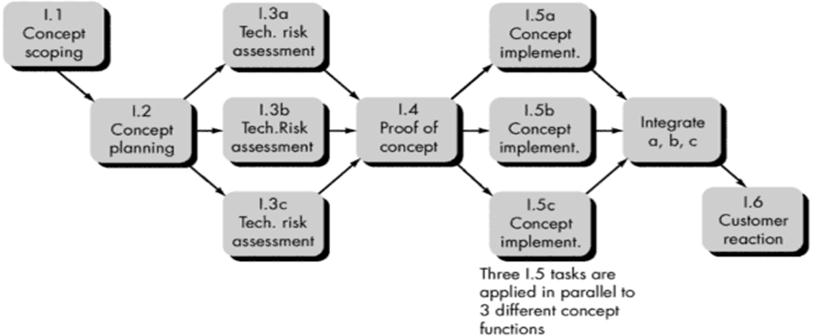
* Major tasks may be used to deﬁne a macroscopic schedule for a project.
* Macroscopic schedule must be reﬁned to create a detailed project schedule.
* Reﬁnement begins by taking each major task and decomposing it into a set of subtasks (with related work products and milestones).

Example of task decomposition - ***concept scoping*** for a development project:

### Deﬁning a Task Network

* Individual tasks and subtasks have interdependencies based on their sequence.
* It is likely that development activities and tasks will be performed in parallel.
* Concurrent tasks must be coordinated so that they will be complete when later tasks require their work product(s).
* ***Task network ( activity network )*** is a graphic representation of the task ﬂow for a project.

o The task network depicts major software engineering tasks.



* Parallel tasks occur asynchronously
* Planner must determine intertask dependencies to ensure continuous progress toward completion.
* Project manager should be aware of those tasks that must be completed on schedule if the project as a whole is to be completed on schedule.

### Scheduling

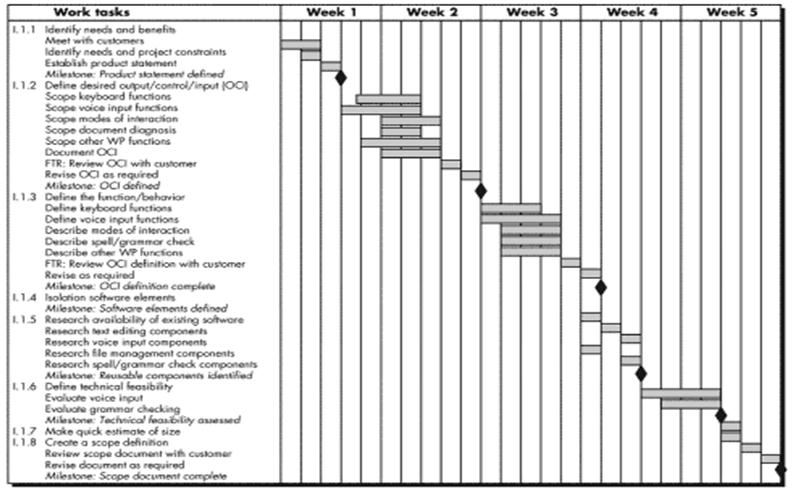
* Generalized project scheduling tools and techniques can be applied with little modiﬁcation to software projects.
* *Program evaluation and review technique* (PERT) and *critical path method* (CPM) are two project scheduling methods that can be applied to software development.
* Both techniques are driven by information already developed in earlier project planning activities:
* Estimates of effort
* A decomposition of the product function
* The selection of the appropriate process model and task set
* Decomposition of tasks
* Interdependencies among tasks may be deﬁned using a task network.
* Tasks (*work breakdown structure* (WBS)) are deﬁned for the product as a whole or for individual functions.
* PERT and CPM provide quantitative tools that allow the software planner to

1. determine the *critical path—*the chain of tasks that determines the duration of the project
2. establish "most likely" time estimates for individual tasks by applying statistical models
3. calculate "boundary times" that deﬁne a time "window" for a particular task.

* Important boundary times discerned from a PERT or CPM network:
  1. the earliest time that a task can begin when all preceding tasks are completed in the shortest possible time
  2. the latest time for task initiation before the minimum project completion time is delayed
  3. the earliest ﬁnish—the sum of the earliest start and the task duration
  4. the latest ﬁnish—the latest start time added to task duration
  5. the *total ﬂoat—*the amount of surplus time or leeway allowed in scheduling tasks so that the network critical path is maintained on schedule.
* Boundary time calculations lead to a determination of critical path and provide the manager with a quantitative method for evaluating progress as tasks are completed.
* PERT and CPM implemented in a variety of automated tools that are available for the personal computer
* Such tools are easy to use and make the scheduling methods described previously available to every software project manager

### Timeline Charts

* Software project schedule - planner begins with a set of tasks
* Effort, duration, and start date are input for each task.
* Tasks may be assigned to speciﬁc individuals.
* ***Timeline chart****,* ( *Gantt chart)* generated.:



Depicts a part of a software project schedule that emphasizes the task for a new word-processing (WP) software product.

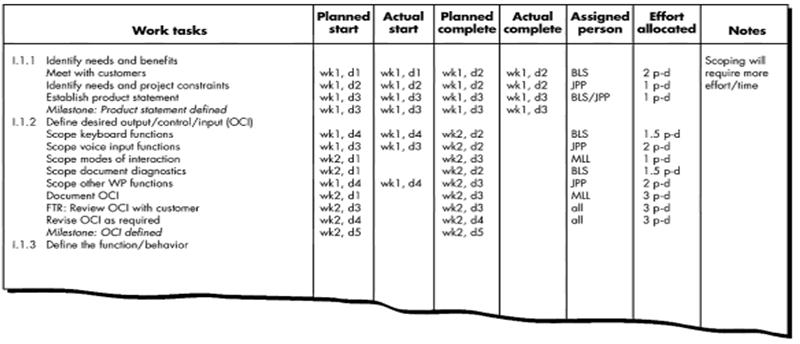
All project tasks are listed in the left-hand column. Horizontal bars indicate the duration of each task.

When multiple bars occur at the same time on the calendar, task concurrency is implied. The diamonds indicate milestones.

Software project scheduling tools produce *project tables—*a tabular listing of all: project tasks

their planned and actual start- and end-dates, variety of related information

Used in conjunction with the timeline chart, project tables enable the project manager to track progress.



### Tracking the Schedule

* The project schedule provides a road map for a software project manager.
* Project schedule deﬁnes the tasks and milestones that must be tracked and controlled as the project proceeds.
* Tracking can be accomplished in a number of different ways:
* Conducting periodic project status meetings in which each team member reports progress and problems.
* Evaluating the results of all reviews conducted throughout the software engineering process.
* Determining whether formal project milestones have been accomplished by the scheduled date.
* Comparing actual start-date to planned start-date for each project task listed in the resource table
* Meeting informally with practitioners to obtain their subjective assessment of progress to date and problems on the horizon.
* *Time-boxing* strategy recognizes that the complete product may not be deliverable by the predeﬁned deadline.
* An incremental software paradigm is chosen and a schedule is derived for each incremental delivery.
* Tasks associated with each increment are time-boxed.
* The schedule for each task is adjusted by working backward from the delivery date for the increment.
* A "box" is put around each task.
* When a task hits the boundary of its time box (plus or minus 10 percent), work stops and the next task begins.

### Earned Value Analysis

* ***Earned value analysis* (EVA)** - qualitative approaches to project tracking
* Provides a common value scale for every software project task, regardless of the type of work being performed.
* The total hours to do the whole project are estimated, and every task is given an earned value based on its

estimated percentage of the total.

* Earned value is a measure of progress

o Able to assess the "percent of completeness" of a project using quantitative analysis

### Following steps performed:

1. The *budgeted cost of work scheduled* (BCWS) is determined for each work task represented in the schedule.
   1. The work (in person-hours or person-days) of each software engineering task is planned.
   2. BCWS***i*** is the effort planned for work task ***i****.*
   3. To determine progress at a given point along the project schedule, the value of BCWS is the sum of the **BCWS*i*** values for all work tasks that should have been completed by that point in time on the project schedule.
2. The BCWS values for all work tasks are summed to derive the budget at completion, BAC. Hence, BAC = S (BCWS*k*) for all tasks ***k***
3. Next, the value for *budgeted cost of work performed* (BCWP) is computed.
   1. The value for BCWP is the sum of the BCWS values for all work tasks that have actually been completed by a point in time on the project schedule.

* The distinction between BCWS and BCWP is that the former represents the budget of the activities that were planned to be completed and the latter represents the budget of the activities that actually were completed.
* Given values for BCWS, BAC, and BCWP, progress indicators can be computed:

Schedule performance index, SPI = BCWP/BCWS Schedule variance, SV = BCWP - BCWS

* SPI is an indication of the eﬃciency with which the project is utilizing scheduled resources.

o An SPI value close to 1.0 indicates eﬃcient execution of the project schedule.

* SV is an absolute indication of variance from the planned schedule.
* Percent scheduled for completion = BCWS/BAC

o provides an indication of the percentage of work that should have been completed by time *t.*

* Percent complete = BCWP/BAC

o provides a quantitative indication of the percent of completeness of the project at a given point in time, *t.*

* *Actual cost of work performed,* ACWP - the sum of the effort expended on work tasks that have been completed by a point in time on the project schedule.
* It is then possible to compute

Cost performance index, CPI = BCWP/ACWP Cost variance, CV = BCWP - ACWP

* CPI close to 1.0 provides a strong indication that the project is within its deﬁned budget.
* CV is an absolute indication of cost savings (against planned costs) or shortfall at a particular stage of a project.

### Error Tracking

* Throughout the software process, a project team errors associated with each work product.
* Error-related measures and resultant metrics are collected over many software projects, can be used as a baseline for comparison against error data collected in real time.
* Error tracking can be used as one means for assessing the status of a current project.

### Defect removal eﬃciency

* Software team performs formal technical reviews (and, later, testing) to ﬁnd and correct errors, ***E****,* in work products produced during software engineering tasks.
* Any errors that are not uncovered (but found in later tasks) are defects, ***D****.*
* Defect removal eﬃciency is deﬁned as

**DRE = *E/*(*E* + *D*)**

* DRE is a process metric that provides a strong indication of the effectiveness of quality assurance activities
* DRE can also be used to assist a project manager in determining the progress that is being made as a software project moves through its scheduled work tasks.

o e.g. a software organization has collected error and defect data over the past 24 months and has developed

averages for the following metrics:

§ Errors per requirements speciﬁcation page, *E*req

§ Errors per component—design level, *E*design

§ Errors per component—code level, *E*code

§ DRE—requirements analysis

§ DRE—architectural design

§ DRE—component level design

§ DRE—coding

* As the project progresses through each software engineering step, the software team records and reports the number of errors found during requirements, design, and code reviews.
* The project manager calculates current values for *E*req, *E*design, and *E*code.
* Compared to averages for past projects.
* If current results vary by more than 20% from the average, there may be cause for concern and there is certainly cause for investigation.
* For example, if *E*req = 2.1 for project X, yet the organizational average is 3.6, one of two scenarios is possible:

1. the software team has done an outstanding job of developing the requirements speciﬁcation or
2. the team has been lax in its review approach. If the second scenario appears likely, the project manager should take immediate steps to build additional design time into the schedule to accommodate the requirements

defects that have likely been propagated into the design activity.

### The Project Plan

* Each step in the software engineering process should produce a deliverable that can be reviewed and that can act as a foundation for the steps that follow.
* The *Software Project Plan* is produced at the culmination of the planning tasks.
* It provides baseline cost and scheduling information that will be used throughout the software process.
* **The *Software Project Plan* must**
  1. communicate scope and resources to software management, technical staff, and the customer
  2. deﬁne risks and suggest risk aversion techniques
  3. deﬁne cost and schedule for management review
  4. provide an overall approach to software development for all people associated with the project

outline how quality will be ensured and change will be ma