UNIT V PROJECT MANAGEMENT

ESTIMATION

- S/W is the most expensive element of virtually all computer based systems
- The accuracy of a s/w project estimate is predicated on a number of things:
    - ✓ The degree to which the planner has properly estimated the size of the product to be built
    - ✓ The ability to translate the size estimate into human effort, calendar time, and dollars (required availability of past records)
    - ✓ The degree to which the project plan reflects the abilities of the s/w team
    - ✓ The stability of product requirements and the environment that supports the s/w engineering effort
- Sizing represents the project planner's first major challenge
- Size refers to a quantifiable outcome of the s/w project (e.g. LOC and/or FP)

Size-Oriented Metrics

- LOC measure claim that LOC is an "artifact" of all software development projects that can be easily counted, that many existing software estimation models use LOC or KLOC as a key input, and that a large body of literature and data predicated on LOC already exists.
- The planner must estimate the LOC to be produced long before analysis and design has been completed.

Function-Oriented Metrics

- Function-oriented software metrics use a measure of the functionality delivered by the application as a normalization value. The most widely used function-oriented metric is the function point (FP). Computation of the function point is based on characteristics of the software's information domain and complexity.
- The function point, like the LOC measure, is controversial. Proponents claim that FP is programming language independent, making it ideal for applications using conventional and nonprocedural languages, and that it is based on data that are more likely to be known early in the evolution of a project, making FP more attractive as an estimation approach.
- Opponents claim that the method requires some "sleight of hand" in that computation is based on subjective rather than objective data, that counts of the information domain (and other dimensions) can be difficult to collect after the fact, and that FP has no direct physical meaning—it's just a number.

A **function point** is a "unit of measurement" to express the amount of business functionality an information system (as a product) provides to a user. Function points measure software size. The cost (in dollars or hours) of a single unit is calculated from past projects.

It can be used to estimate

i.the cost or effort required to design, code and test the software.

ii.predict the number of errors that will be encountered during testing

iii.forecast the number of components and/or the number of projected source lines in the implemented system.

- **External inputs:** those items provided by the user that describe distinct application-oriented data (such as file names and menu selections). Enquiries are counted separately. Inputs used to update internal logical files(ILFs)
- **External outputs:** those items provided to the user that generate distinct application-oriented data (such as reports and messages).
- **External inquiries:** interactive inputs requiring a response.
- **External files:** machine-readable interfaces to other systems.
- **Internal files:** logical master files in the system.

$$FP=\text{count total} *[0.65+0.01* \sum (Fi)]$$

Where count total is the sum of all FP entries obtained from the below table.

Information Domain Value	Count		Weighting factor Simple	Average	Complex		
External Inputs (EIs)		×	3	4	6	=	
External Outputs (EOs)		×	4	5	7	=	
External Inquiries (EQs)		×	3	4	6	=	
Internal Logical Files (ILFs)		×	7	10	15	=	
External Interface Files (EIFs)		×	5	7	10	=	
Count total							

The Fi (I 1 t0 14) are valued adjustment factors based on responses to the following questions

1. Does the system require reliable backup and recovery?
2. Are specialized data communications required to transfer information to or from the application?
3. Are there distributed processing functions?
4. Is performance critical?
5. Will the system run I existing ,heavily utilized operational environment ?
6. Does the system require online data entry?
7. Does the online data entry require the input transaction to be built over multiple screens or operations?
8. Are the ILFs updated online?
9. Are the inputs,outputs,files or inquiries complex?
10. Is the internal processing complex?
11. Is the code designed to be reusable?

12. Are conversion and installation included in the design?
13. IS the system designed for multiple installations in different organizations?
14. Is the application designed to facilitate change and ease of use by the user?

Reconciling LOC and FP Metrics

The relationship between lines of code and function points depends upon the programming language that is used to implement the software and the quality of the design. Function points and LOC-based metrics have been found to be relatively accurate predictors of software development effort and cost. However, in order to use LOC and FP for estimation, an historical baseline of information must be established. Within the context of process and project metrics, productivity and quality should be concerned which are the measures of software development "output" as a function of effort and time applied and measures of the "fitness for use" of the work products that are produced. For process improvement and project planning purposes, the interest is historical.

Make/Buy Decision

- It is often more cost effective to acquire rather than develop software
- Managers have many acquisition options
  - Software may be purchased (or licensed) off the shelf
  - "Full-experience" or "partial-experience" software components may be acquired and integrated to meet specific needs
  - Software may be custom built by an outside contractor to meet the purchaser's specifications
- The make/buy decision can be made based on the following conditions
  - Will the software product be available sooner than internally developed software?
  - Will the cost of acquisition plus the cost of customization be less than the cost of developing the software internally?
  - Will the cost of outside support (e.g., a maintenance contract) be less than the cost of internal support?

COCOMO II

Barry Boehm introduced COCOMO II (COst COnstructive MOdel) which is an hierarchy of estimation models that addresses the following areas:

- Application composition model. Used during the early stages of software engineering, when prototyping of user interfaces, consideration of software and system interaction, assessment of performance, and evaluation of technology maturity are paramount.
- Early design stage model. Used once requirements have been stabilized and basic software architecture has been established.
- Post-architecture-stage model. Used during the construction of the software.

Three different sizing options are available as part of the model hierarchy: object points, function points, and lines of source code.

OBJECT TYPE	COMPLEXITY WEIGHT		
	SIMPLE	MEDIUM	DIFFICULT
Screen	1	2	3
Report	2	5	8
3GL component			10

The object point is an indirect software measure that is computed using counts of the number of (1) screens (at the user interface), (2) reports, and (3) components likely to be required to build the application.

When component-based development or general software reuse is to be applied, the percent of reuse (%reuse) is estimated and the object point count is adjusted:

$$NOP=(\text{Object points})*[(100-\%\text{reuse})/100]$$

where NOP is defined as new object points.

To derive an estimate of effort based on the computed NOP value, a "productivity rate" must be derived.

$$PROD=\frac{NOP}{\text{Person-month}}$$

for different levels of developer experience and development environment maturity.

Once the productivity rate has been determined, an estimate of project effort is computed using

$$\text{Estimated effort}=\frac{NOP}{PROD}$$

In more advanced COCOMO II models, 12 a variety of scale factors, cost drivers, and adjustment procedures are required.

Developer's experience/capability	Very low	Low	Nominal	High	Very high
Environment maturity/capability	Very low	Low	Nominal	High	Very high
PROD	4	7	13	25	50

PROJECT PLANNING PROCESS

The objective of software project planning is to provide a framework that enables the manager to make reasonable estimates of resources, cost, and schedule.

The plan must be adapted and updated as the project proceeds.

Task Set for Project Planning

1. Establish project scope.
2. Determine feasibility.
3. Analyze risks (Chapter 28).
4. Define required resources.

a. Determine required human resources.
b. Define reusable software resources.
c. Identify environmental resources.

5. Estimate cost and effort.
   a. Decompose the problem.
   b. Develop two or more estimates using size, function points, process tasks, or use cases.
   c. Reconcile the estimates.
6. Develop a project schedule
   a. Establish a meaningful task set.
   b. Define a task network.
   c. Use scheduling tools to develop a time-line chart.
   d. Define schedule tracking mechanisms.

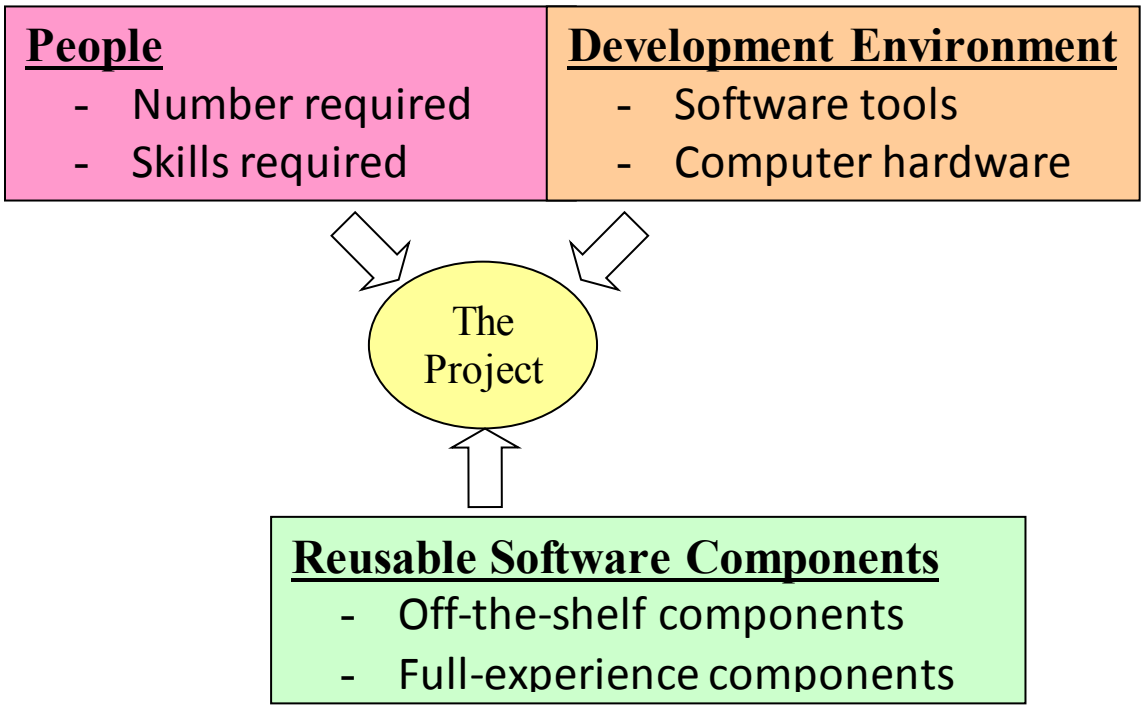
Scope and Feasibility

- Software scope describes
  - The functions and features that are to be delivered to end users
  - The data that are input to and output from the system
  - The "content" that is presented to users as a consequence of using the software
  - The performance, constraints, interfaces, and reliability that bound the system
- Scope can be define using two techniques
  - A narrative description of software scope is developed after communication with all stakeholders
  - A set of use cases is developed by end users
- After the scope has been identified, two questions are asked
  - Can we build software to meet this scope?
  - Is the project feasible?
- Software engineers too often rush (or are pushed) past these questions
- Later they become mired in a project that is doomed from the onset
- After the scope is resolved, feasibility is addressed
- Software feasibility has four dimensions
  - Technology – Is the project technically feasible? Is it within the state of the art? Can defects be reduced to a level matching the application's needs?
  - Finance – Is is financially feasible? Can development be completed at a cost that the software organization, its client, or the market can afford?
  - Time – Will the project's time-to-market beat the competition?
  - Resources – Does the software organization have the resources needed to succeed in doing the project?

Project Resources

- Three major categories of software engineering resources
  - People
  - Development environment
  - Reusable software components
    - Often neglected during planning but become a paramount concern during the construction phase of the software process
- Each resource is specified with
  - A description of the resource
  - A statement of availability
  - The time when the resource will be required
  - The duration of time that the resource will be applied

Categories of Resources



Human Resources

- Planners need to select the number and the kind of people skills needed to complete the project
- They need to specify the organizational position and job specialty for each person
- Small projects of a few person-months may only need one individual
- Large projects spanning many person-months or years require the location of the person to be specified also
- The number of people required can be determined only after an estimate of the development effort

Development Environment Resources

- A software engineering environment (SEE) incorporates hardware, software, and network resources that provide platforms and tools to develop and test software work products
- Most software organizations have many projects that require access to the SEE provided by the organization
- Planners must identify the time window required for hardware and software and verify that these resources will be available

Reusable Software Resources

- Off-the-shelf components
  - Components are from a third party or were developed for a previous project
  - Ready to use; fully validated and documented; virtually no risk
- Full-experience components
  - Components are similar to the software that needs to be built
  - Software team has full experience in the application area of these components
  - Modification of components will incur relatively low risk
- Partial-experience components
  - Components are related somehow to the software that needs to be built but will require substantial modification
  - Software team has only limited experience in the application area of these components
  - Modifications that are required have a fair degree of risk
- New components
  - Components must be built from scratch by the software team specifically for the needs of the current project
  - Software team has no practical experience in the application area
  - Software development of components has a high degree of risk

RISK MANAGEMENT

Risk concerns with failure happenings. Risk involves change, such as in changes in mind,
opinion, actions, or places.

- uncertainty—the risk may or may not happen;
- loss—if the risk becomes a reality, unwanted consequences or losses will occur

Project risks threaten the project plan. That is, if project risks become real, it is likely that the project schedule will slip and that costs will increase. Project risks identify potential budgetary, schedule, personnel (staffing and organization), resource, stakeholder, and requirements problems and their impact on a software project.

Technical risks threaten the quality and timeliness of the software to be produced. If a technical risk becomes a reality, implementation may become difficult or impossible. Technical risks identify potential design, implementation, interface, verification, and maintenance problems.

Business risks threaten the viability of the software to be built and often jeopardize the project or the product. Candidates for the top five business risks are

- building an excellent product or system that no one really wants (market risk),
- building a product that no longer fits into the overall business strategy for the company (strategic risk),
- building a product that the sales force doesn't understand how to sell (sales risk),
- losing the support of senior management due to a change in focus or a change in people (management risk), and
- losing budgetary or personnel commitment (budget risks).

Known risks are those that can be uncovered after careful evaluation of the project plan, the business and technical environment in which the project is being developed, and other reliable information sources (e.g., unrealistic delivery date, lack of documented requirements or software scope, poor development environment).

Predictable risks are extrapolated from past project experience (e.g., staff turnover, poor communication with the customer, dilution of staff effort as ongoing maintenance requests are serviced). Unpredictable risks are the joker in the deck. They can and do occur, but they are extremely difficult to identify in advance.

Risk Identification

Risk identification is a systematic attempt to specify threats to the project plan (estimates, schedule, resource loading, etc.). By identifying known and predictable risks, the project manager takes a first step toward avoiding them when possible and controlling them when necessary.

There are two distinct types of risks for each of the categories: generic risks and product-specific risks.

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

One method for identifying risks is to create a risk item checklist. The checklist can be used for risk identification and focuses on some subset of known and predictable risks in the following generic subcategories:

- Product size—risks associated with the overall size of the software to be built or modified.
- Business impact—risks associated with constraints imposed by management or the marketplace.
- Stakeholder characteristics—risks associated with the sophistication of the stakeholders and the developer's ability to communicate with stakeholders in a timely manner.
- Process definition—risks associated with the degree to which the software process has been defined and is followed by the development organization.
- Development environment—risks associated with the availability and quality of the tools to be used to build the product.
- Technology to be built—risks associated with the complexity of the system to be built and the "newness" of the technology that is packaged by the system.
- Staff size and experience—risks associated with the overall technical and project experience of the software engineers who will do the work.

Risk Projection

Risk projection, also called risk estimation, attempts to rate each risk in two ways— the likelihood or probability that the risk is real and the consequences of the problems associated with the risk, should it occur. You work along with other managers and technical staff to perform four risk projection steps:

- Establish a scale that reflects the perceived likelihood of a risk.
- Delineate the consequences of the risk.
- Estimate the impact of the risk on the project and the product.
- Assess the overall accuracy of the risk projection so that there will be no misunderstandings.

A risk table provides you with a simple technique for risk projection.

The overall risk exposure RE is determined using the following relationship:

$$RE = P * C$$

Risk Refinement

One way to do this is to represent the risk in condition-transition-consequence (CTC) format. That is, the risk is stated in the following form:

Given that <condition> then there is concern that (possibly) <consequence>.

This general condition can be refined in the following manner:

Subcondition 1. Certain reusable components were developed by a third party with no knowledge of internal design standards.

Subcondition 2. The design standard for component interfaces has not been solidified and may not conform to certain existing reusable components.

Subcondition 3. Certain reusable components have been implemented in a language that is not supported on the target environment.

Risk Mitigation,Monitoring and Management

An effective strategy must consider three issues: risk avoidance, risk monitoring, and risk management and contingency planning.

To mitigate this risk, a strategy has to be developed for reducing turnover. Among the possible steps to be taken are:

- Meet with current staff to determine causes for turnover (e.g., poor working conditions, low pay, competitive job market).
- Mitigate those causes that are under your control before the project starts.
- Once the project commences, assume turnover will occur and develop techniques to ensure continuity when people leave.
- Organize project teams so that information about each development activity is widely dispersed.
- Define work product standards and establish mechanisms to be sure that all models and documents are developed in a timely manner.
- Conduct peer reviews of all work (so that more than one person is "up to speed").
- Assign a backup staff member for every critical technologist.

PROJECT SCHEDULING AND TRACKING

Software project scheduling is an action that distributes estimated effort across the planned project duration by allocating the effort to specific software engineering tasks. Scheduling for software engineering projects can be viewed from two rather different perspectives. In the first, an end date for release of a computer-based system has already (and irrevocably) been established. The software organization is constrained to distribute effort within the prescribed time frame. The second view of
software scheduling assumes that rough chronological bounds have been discussed but that the end date is set by the software engineering organization.

Basic Principles

- **Compartmentalization.** The project must be compartmentalized into a number of manageable activities and tasks. To accomplish compartmentalization, both the product and the process are refined.
- **Interdependency.** The interdependency of each compartmentalized activity or task must be determined. Some tasks must occur in sequence, while others can occur in parallel. Some activities cannot commence until the work product produced by another is available. Other activities can occur independently.
- **Time allocation.** Each task to be scheduled must be allocated some number of work units (e.g., person-days of effort). In addition, each task must be 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 people on the software team. As time allocation occurs, you must ensure that no more than the allocated number of people has been scheduled at any given time. For example, consider a project that has three assigned software engineers (e.g., three person-days are available per day of assigned effort4). On a given day, seven concurrent tasks

must be accomplished. Each task requires 0.50 person-days of effort. More effort has been allocated than there are people to do the work.

- **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 (e.g., the design of a component) or a part of a work product. Work products are often combined in deliverables.
- **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.

The Relationship Between People and Effort

L, is related to effort and development time by the equation:

$$L = P * E^{1/3} t^{4/3}$$

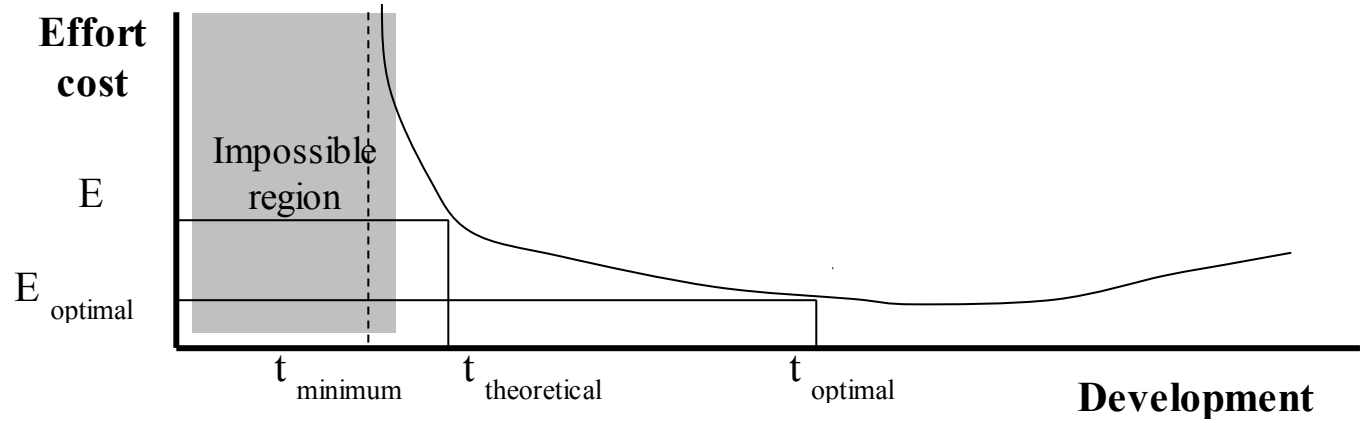
Rearranging this software equation, we can arrive at an expression for development effort *E*:

$$E = \frac{L^3}{P^3 t^4}$$

Effort Distribution

A recommended distribution of effort across the software process is often referred to as the 40–20–40 rule. Forty percent of all effort is allocated to frontend analysis and design. A similar percentage is applied to back-end testing. 20 percent of effort is deemphasized in the coding.

Because of the effort applied to software design, code should follow with relatively little difficulty. A range of 15 to 20 percent of overall effort can be achieved. Testing and subsequent debugging can account for 30 to 40 percent of software development effort. The criticality of the software often dictates the amount of testing that is required. If software is human rated (i.e., software failure can result in loss of life), even higher percentages are typical.



Task Network

Task Set

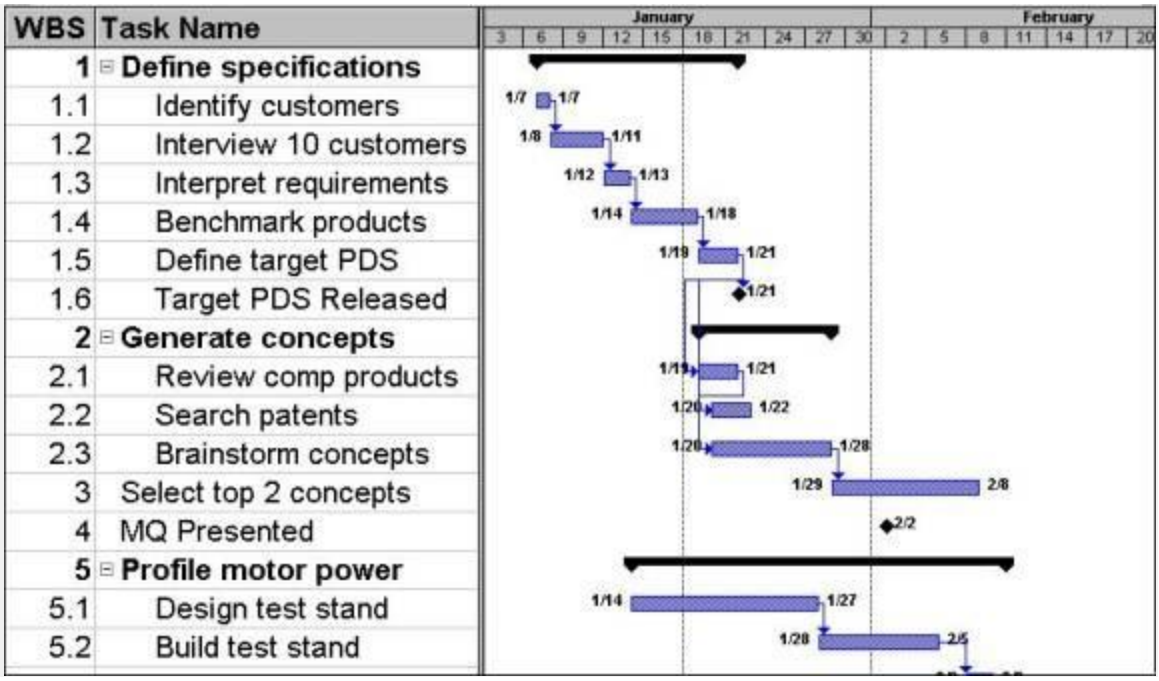
- A task set is the work breakdown structure for the project
- No single task set is appropriate for all projects and process models
  - It varies depending on the project type and the degree of rigor (based on influential factors) with which the team plans to work
- The task set should provide enough discipline to achieve high software quality
  - But it must not burden the project team with unnecessary work

Purpose of a Task Network

- Also called an activity network
- It is a graphic representation of the task flow for a project
- It depicts task length, sequence, concurrency, and dependency
- Points out inter-task dependencies to help the manager ensure continuous progress toward project completion
- The critical path
  - ✓ A single path leading from start to finish in a task network
  - ✓ It contains the sequence of tasks that must be completed on schedule if the project as a whole is to be completed on schedule
  - ✓ It also determines the minimum duration of the project

Timeline Chart

- Also called a Gantt chart; invented by Henry Gantt, industrial engineer, 1917
- All project tasks are listed in the far left column
- The next few columns may list the following for each task: projected start date, projected stop date, projected duration, actual start date, actual stop date, actual duration, task inter-dependencies (i.e., predecessors)
- To the far right are columns representing dates on a calendar
- The length of a horizontal bar on the calendar indicates the duration of the task
- When multiple bars occur at the same time interval on the calendar, this implies task concurrency
- A diamond in the calendar area of a specific task indicates that the task is a milestone; a milestone has a time duration of zero



Project Table

Work tasks	Planned start	Actual start	Planned complete	Actual complete	Assigned person	Effort allocated	Notes
I.1.1 Identify needs and benefits							Scoping will require more effort/time
Meet with customers	wk1, d1	wk1, d1	wk1, d2	wk1, d2	BLS	2 p-d	
Identify needs and project constraints	wk1, d2	wk1, d2	wk1, d2	wk1, d2	JPP	1 p-d	
Establish product statement	wk1, d3	wk1, d3	wk1, d3	wk1, d3	BLS/JPP	1 p-d	
Milestone: Product statement defined	wk1, d3	wk1, d3	wk1, d3	wk1, d3			
I.1.2 Define desired output/control/input (OCI)							
Scope keyboard functions	wk1, d4	wk1, d4	wk2, d2		BLS	1.5 p-d	
Scope voice input functions	wk1, d3	wk1, d3	wk2, d2		JPP	2 p-d	
Scope modes of interaction	wk2, d1		wk2, d3		MLL	1 p-d	
Scope document diagnostics	wk2, d1		wk2, d2		BLS	1.5 p-d	
Scope other WP functions	wk1, d4	wk1, d4	wk2, d3		JPP	2 p-d	
Document OCI	wk2, d1		wk2, d3		MLL	3 p-d	
FTR: Review OCI with customer	wk2, d3		wk2, d3		all	3 p-d	
Revise OCI as required	wk2, d4		wk2, d4		all	3 p-d	
Milestone: OCI defined	wk2, d5		wk2, d5				
I.1.3 Define the function/behavior							

Earned value analysis

The earned value system 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.

To determine the earned value, the following steps are performed:

- The budgeted cost of work scheduled (BCWS) is determined for each work task represented in the schedule.

- The BCWS values for all work tasks are summed to derive the budget at completion (BAC). Hence,

$$BAC = \sum(BCWS_k) \text{ for all task } k$$

- Next, the value for budgeted cost of work performed (BCWP) is computed. 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.

Given values for BCWS, BAC, and BCWP, important progress indicators can be computed:

$$\text{Schedule performance index, SPI} = \frac{BCWP}{BCWS}$$

$$\text{Schedule Variance, SV} = BCWP - BCWS$$

$$\text{Percent scheduled for completion} = \frac{BCWS}{BAC}$$

$$\text{Percent complete} = \frac{BCWP}{BAC}$$

$$\text{Cost performance index, CPI} = \frac{BCWP}{ACWP}$$

$$\text{Cost variance, CV} = BCWP - ACWP$$

A CPI value 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. Like over-the-horizon radar, earned value analysis illuminates scheduling difficulties before they might otherwise be apparent. This enables you to take corrective action before a project crisis develops.

PROCESS AND PROJECT METRICS-DECOMPOSITION TECHNIQUES

The decomposition approach was discussed from two different points of view: decomposition of the problem and decomposition of the process.

Software Sizing

The accuracy of a software project estimate is predicated on a number of things:

- the degree to which you have properly estimated the size of the product to be built;
- the ability to translate the size estimate into human effort, calendar time, and dollars
  (a function of the availability of reliable software metrics from past projects);
- the degree to which the project plan reflects the abilities of the software team; and
- the stability of product requirements and the environment that supports the software engineering effort.

Problem-Based Estimation

LOC and FP data are used in two ways during software project estimation: (1) as estimation variables to "size" each element of the software and (2) as baseline metrics collected from past projects and used in conjunction with estimation variables to develop cost and effort projections.

There is often substantial scatter in productivity metrics for an organization, making the use of a single-baseline productivity metric suspect. In general, LOC/pm or FP/pm averages should be computed by project domain. That is, projects should be grouped by team size, application area, complexity, and other relevant parameters. Local domain averages should then be computed. When a new project is estimated, it should first be allocated to a domain, and then the appropriate domain average for past productivity should be used in generating the estimate.

The LOC and FP estimation techniques differ in the level of detail required for decomposition and the target of the partitioning. When LOC is used as the estimation variable, decomposition is absolutely essential and is often taken to considerable levels of detail. The greater the degree of partitioning, the more likely reasonably accurate estimates of LOC can be developed.

For FP estimates, decomposition works differently. Rather than focusing on function, each of the information domain characteristics—inputs, outputs, data files, inquiries, and external interfaces—as well as the 14 complexity adjustment values are estimated. The resultant estimates can then be used to derive an FP value that can be tied to past data and used to generate an estimate. Regardless of the estimation variable that is used, you should begin by estimating a range of values for each function or information domain value. Using historical data or (when all else fails) intuition, estimate an optimistic, most likely, and pessimistic size value for each function or count for each information domain value. An implicit indication of the degree of uncertainty is provided when a range
of values is specified.

A three-point or expected value can then be computed. The *expected value* for the estimation variable (size) S can be computed as a weighted average of the optimistic (s_{opt}), most likely (s_m), and pessimistic (s_{pess}) estimates. For example

$$S = \frac{S_{opt} + 4S_m + S_{pess}}{6}$$

gives heaviest credence to the "most likely" estimate and follows a beta probability distribution.

Process-based Estimation

Process-based estimation begins with a delineation of software functions obtained from the project scope. A series of framework activities must be performed for each function. Costs and effort for each function and framework activity are computed as the last step. If process-based estimation is performed independently of LOC or FP estimation, two or three estimates for cost and effort are considered that may be compared and reconciled. If both sets of estimates show rea sonable agreement, there is good reason to

believe that the estimates are reliable. If, on the other hand, the results of these decomposition techniques show little agreement, further investigation and analysis must be conducted.(LOC and FP-based estimations, process and problem-based estimations examples has to be provided.

Estimation with Use Cases

- Use cases are described using many different formats and styles—there is no standard form.
- Use cases represent an external view (the user's view) of the software and can therefore be written at many different levels of abstraction.
- Use cases do not address the complexity of the functions and features that are described.
- Use cases can describe complex behaviour (e.g., interactions) that involve many functions and features.

Reconciling Estimates

The estimation techniques discussed in the preceding sections result in multiple estimates that must be reconciled to produce a single estimate of effort, project duration, or cost.

- the scope of the project is not adequately understood or has been misinterpreted by the planner, or
- productivity data used for problem-based estimation techniques is inappropriate for the application, obsolete (in that it no longer accurately reflects the software engineering organization), or has been misapplied.