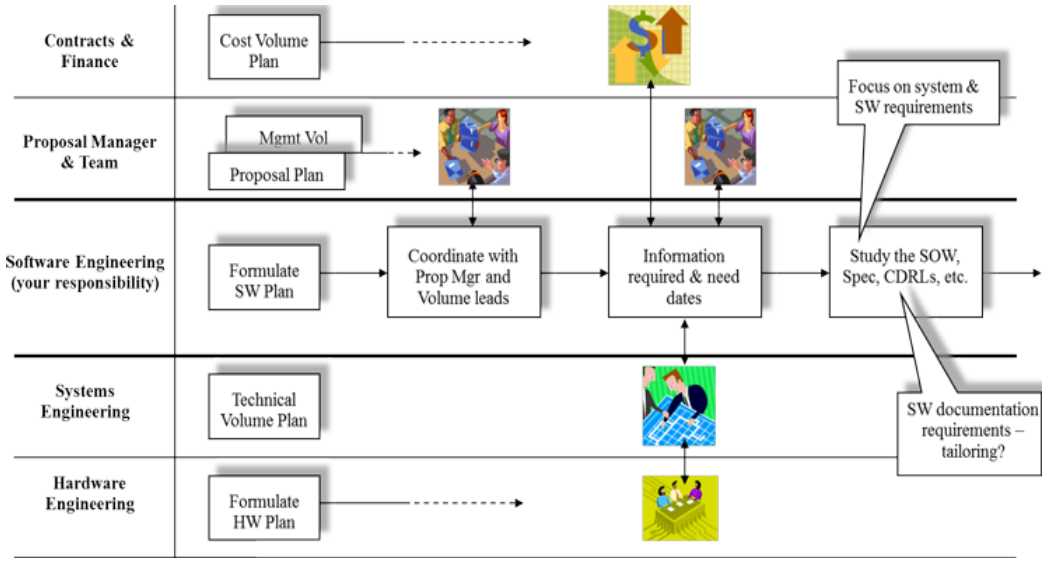
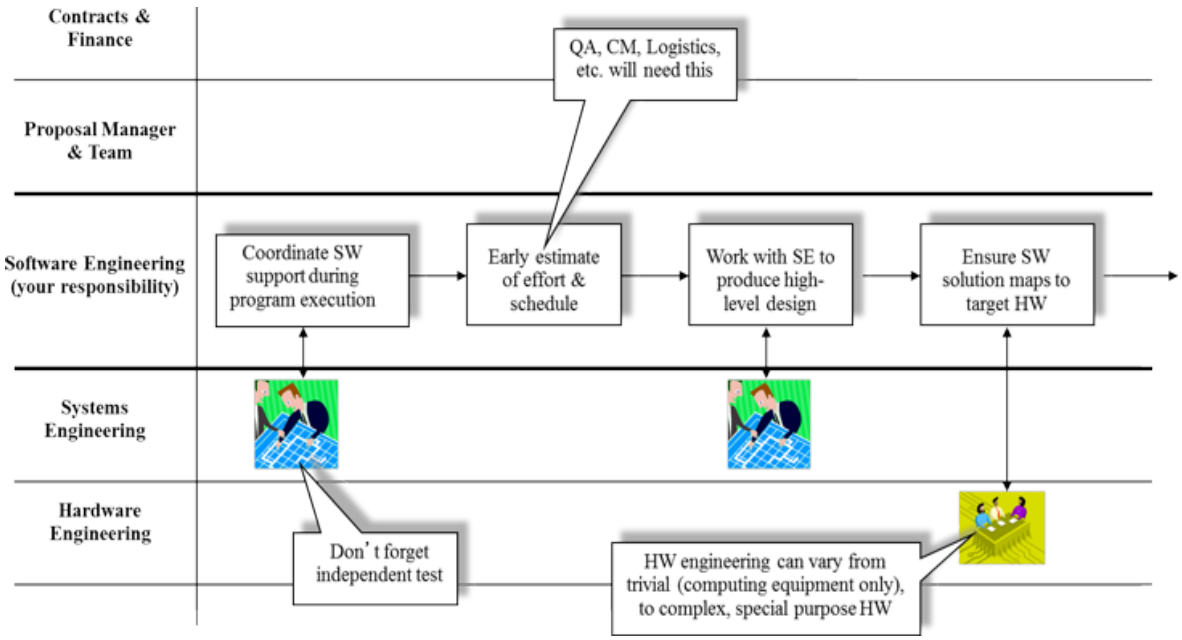
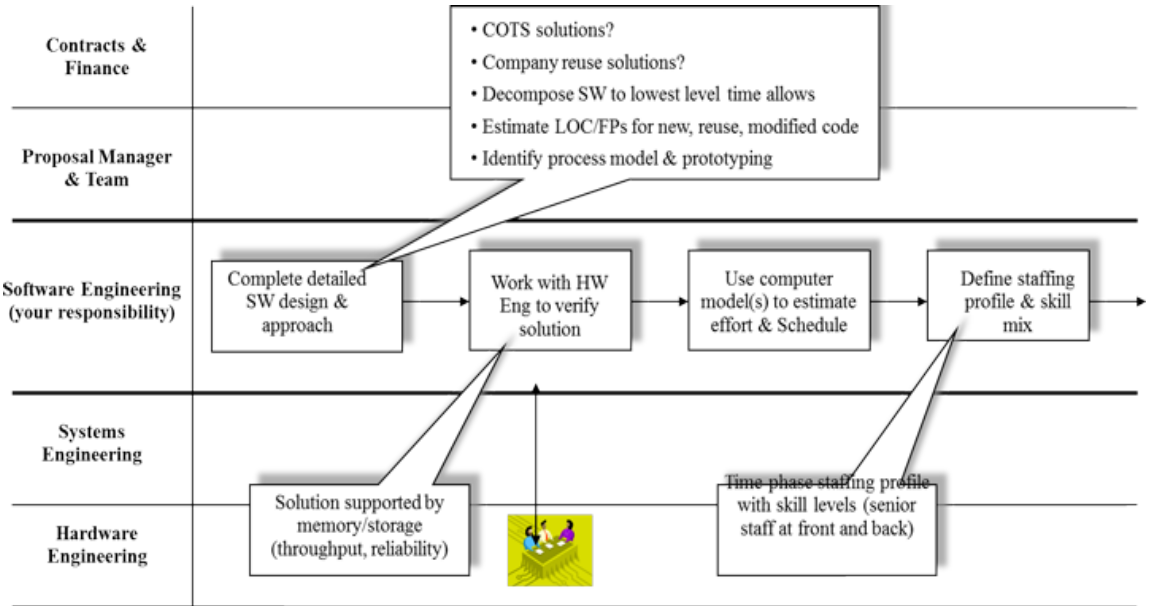
Process Elements

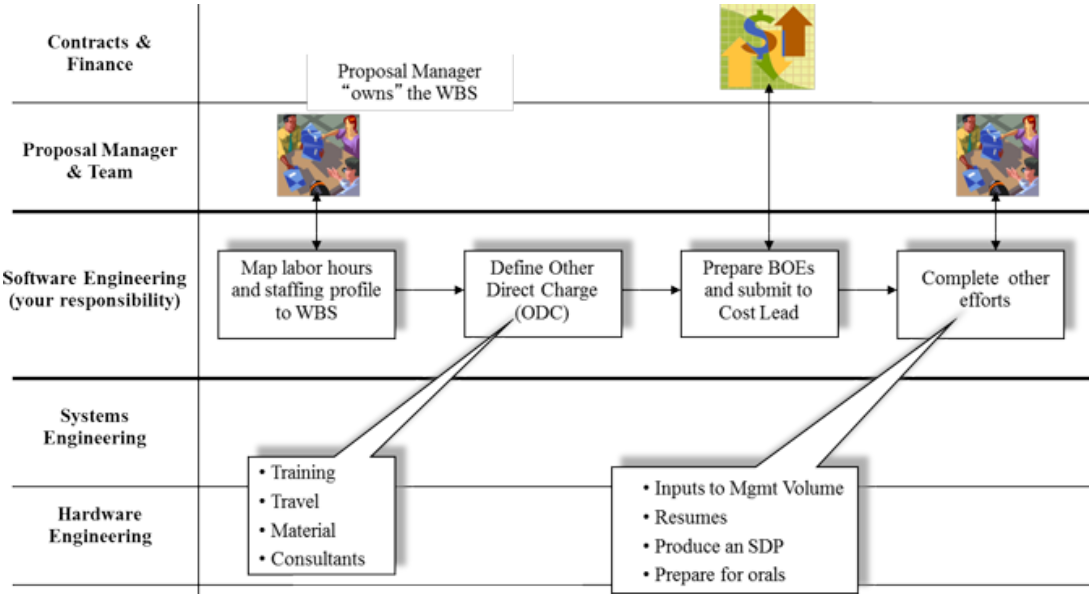
Who is Responsible for Estimating?

As you follow the software engineering responsibilities when developing the software estimate for a proposal, you see how the software manager must constantly coordinate with the other leaders to ensure that the software is planned according to the overall program objective.



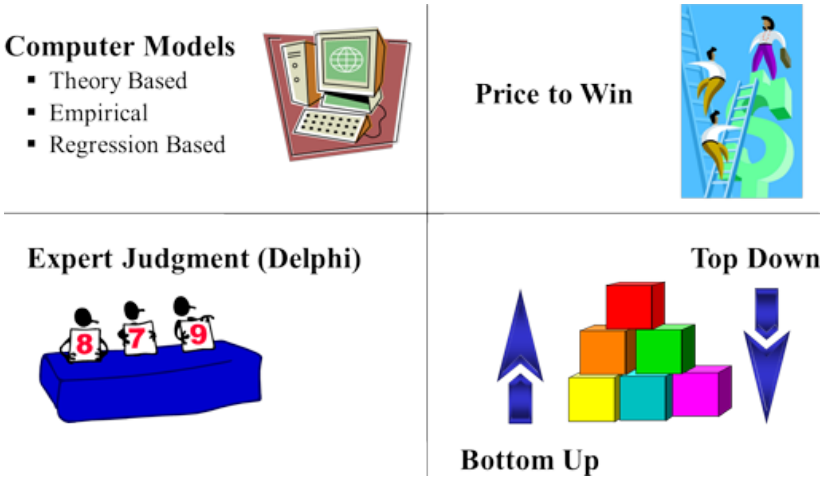



Contracts & Finance
Proposal Manager & Team
Software Engineering (your responsibility)
Systems Engineering
Hardware Engineering
• COTS solutions?
• Company reuse solutions?
• Decompose SW to lowest level time allows
• Estimate LOC/FPs for new, reuse, modified code
• Identify process model & prototyping
Complete detailed SW design & approach
Work with HW Eng to verify solution
Use computer model(s) to estimate effort & Schedule
Define staffing profile & skill mix
Solution supported by memory/storage (throughput, reliability)
Time-phase staffing profile with skill levels (senior staff at front and back)


Contracts & Finance
Proposal Manager & Team
Software Engineering (your responsibility)
Systems Engineering
Hardware Engineering
Proposal Manager "owns" the WBS
Map labor hours and staffing profile to WBS
Define Other Direct Charge (ODC)
Prepare BOEs and submit to Cost Lead
Complete other efforts
• Training
• Travel
• Material
• Consultants
• Inputs to Mgmt Volume
• Resumes
• Produce an SDP
• Prepare for orals

Software Estimation Methods

Software estimation is an art, yet the goal is to make it a science with specific rules, guidelines, procedures, and processes. A variety of methods help estimate the size of software including:



Often project managers and developers will apply multiple methods and compare and contrast the results to help determine the best estimate.

Comparison of Software Estimation Methods

Estimation Method	Description	Advantages	Disadvantages
Regression-based models	Historical information is used to develop algorithms which relate cost with one or more software metrics producing a scatter diagram Examples: COCOMO, REVIC	• Objective, repeatable, easy to use • Can be calibrated to company or project environment	• Inputs may be subjective (need LOCs) • May not handle exceptional circumstances • May be based on inefficient past practices
Empirical-based models	Estimate is based on analogy with cost of previously completed projects in the same domain Example: Checkpoint	• Based on actual experience • Can break down cost at detailed level	• Not always clear how to compare projects • May miss differences between project applications and environments

Theory-based models	Estimate is based on underlying theoretical considerations for software development processes Examples: SLIM, Price-S	• Repeatable • Get what you pay for • Lots of research	• Proprietary • Expensive
Expert judgment (Delphi)	Uses one or more experts to arrive at consensus estimate	• Can factor in differences between past projects and this project • Can factor in exceptions	• Only as good as the experts • Not repeatable
Price-to-win estimate	Estimate is based on whatever the customer has to spend	• Often wins the contract	• Schedule and budget are unrealistic • Engineers become demoralized
Top down estimate	Derive estimate from global properties of the product divided among components	• System level focus will not leave out system level functions	• Does not identify low level technical issues • Sometimes misses detailed components • Does not provide details for cost analysis
Bottom up estimate	Cost of each component is estimated, then costs are added for overall estimation	• Estimate for each component based on detailed understanding • Estimate backed up by personal commitment of individuals • Estimation errors balance out	• Can underestimate by overlooking system level costs • Requires more effort • Some cost elements may be included more than once

As time allows, program/project manager use a variety of techniques including LOC estimation, Function or Feature Points, panel displays, computer models, Delphi method, Price to Win, and Top Down and Bottom Up methods to estimate the cost and schedule.

For more information on software estimation tools and processes, contact:

Service or Tool	Contact Information
COCOMO (COSTAR)	Softstar Systems P.O. Box 1360, Amherst, NH 03031 603-672-0987 http://www.softstarsystems.com/
Function Points	International Function Point Users Group

Service or Tool	Contact Information
	191 Clarksville Road, Princeton Junction, NJ 08550 609-799-4900 http://www.ifpug.org/
PRICE-S	PRICE Systems, L.L.C. 17000 Commerce Parkway, Suite A Mount Laurel, NJ 08054 800-43-PRICE (77423) http://www.pricesystems.com/
SEER	Galorath Incorporated 100 North Sepulveda Blvd, MS 1801, El Segundo, CA 90245 310-414-3220 http://www.galorath.com/
SLIM Tools	Quantitative Software Management (QSM), Inc. 2000 Corporate Ridge, Suite 700, McLean, VA 22102 703-790-0055 http://www.qsm.com/
Process Improvement	Software Engineering Institute, Carnegie Mellon University 4500 Fifth Avenue, Pittsburgh, PA 15213 412-268-5800 http://www.sei.cmu.edu/
Process Improvement	Software Technology Support Center, Ogden Air Logistics Center Hill AFB, UT 84056-5205 801-775-5555 http://www.stsc.hill.af.mil/

Planning a Software Development Effort

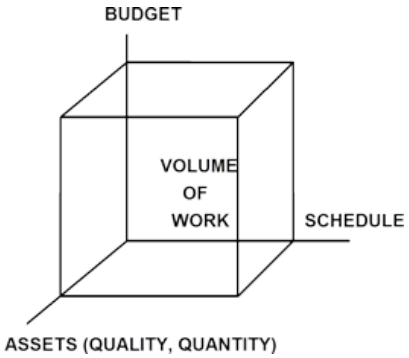
Defining an approach to planning and documenting a software development project is critical to the project's success. You must establish goals that are achievable and recognize that the total volume of work is the result of budget, schedule, and assets (people and equipment) which have both quality and quantity attributes. This module will provide guidance and suggestions for creating and instituting the planning activities. Here are a few quotes to think about:

"Plans are nothing; planning is everything." — *Dwight D. Eisenhower*

"Plans are of little importance, but planning is essential." — *Winston Churchill*

"No battle plan survives contact with the enemy." — *Helmuth von Moltke the Elder*

"A good plan, violently executed now, is better than a perfect plan next week." — *George S. Patton*



It is important to plan for success and ensure:

- Managers have learned how to do effective planning.
- There is a mechanism in the planning process to recognize and create realistic forecasts or predictions.
- Goals are specifically set with its relationship to need, means, and feasibility. There should be a profound knowledge of the organization's business.
- The vision of the project's future is shared and not limited to management.
- Planning is recognized as an on-going process, not an event.
- Planning is done with adequate factual data including customer requirements.
- Do not place too much emphasis on historical data; if may not necessarily be germane to the future.
- Planning must be done as a team, not by a separate planning department or group. People should not want to plan their own destiny.
- Planned activities must be carefully reviewed on an on-going basis.
- Plans and/or activities are reviewed and the process is not punitive. Real facts must be honestly reported and the organization uses this data to succeed and learn. Aggressive planning leads to failures.

What is a Software Development Plan (SDP)

The SDP or Project Plan is a comprehensive planning document that the software manager uses to direct the software development and/or maintenance. The document informs internal management as to the procedures used to manage and organize the software engineering related activities. The SDP is project specific; one must be developed for each project and usually it is a contract requirement. Lastly, the acquisition manager and customer use the SDP to maintain insight into the software development process and activities. While the differences may seems subtle, software Managers distinguish between plans, procedures, guidebooks, and project notebooks as follows:

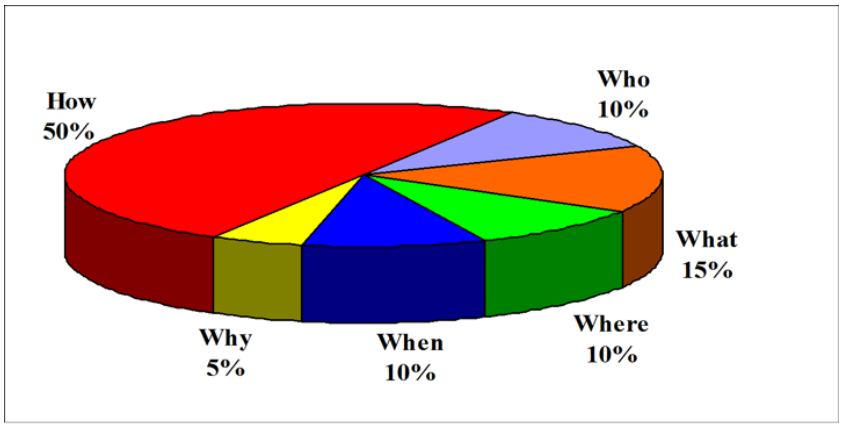
A **plan** is an ordered definition of the "who, what, when, and where." It defines a written strategy, the "why," and may include the tactics or "how" the tasks will be accomplished.

A **procedure** defines how you do something and may include the "who, what, when, and where."

A **guidebook** is a reference set of generic procedures or best practice guidance.

Lastly, a **project notebook** establishes the detailed standards, procedures, guidelines, and restrictions that developers will follow to develop the software; it supplements the SDP and must be in concert with the Project Management Plan.

Empirical data suggests that the approximate division of the elements of planning in the SDP are as follows:



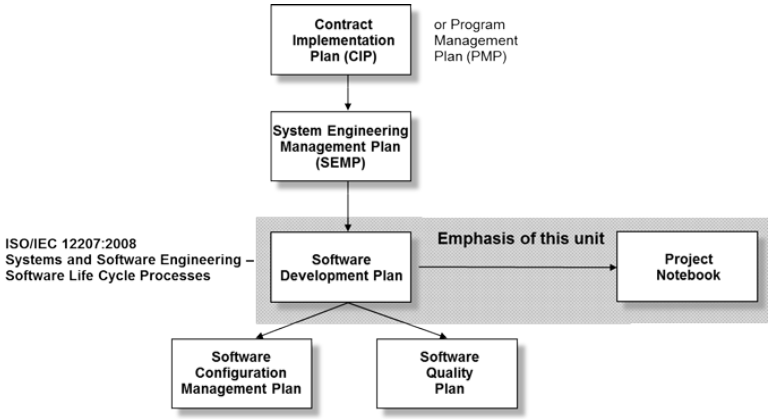
Who (10%)	What (15%)	Where (10%)	When (10%)	Why (5%)	How (50%)
• Staffing • Organization Charts • Roles / Responsibilities	• Tailoring • CSCIs • Activities • Products	• Facilities • Development Environment • Test Environment	• Schedule • Process Model • Milestones	• SOW Compliant • Company Standards	• Process Model • Methodology • Company Standards • Tools

Program/Project staffing, organization charts with functions, and the roles and responsibilities answer **Who** and account for about 10% of the planning document. Generally you would not include an organizational chart with the actual names. Since the Software Development Plan is a deliverable document to your customer in most cases, every time it changes, it would need to be redelivered, so most organizations keep a function organization chart in the document but an organization chart with the team members' names in the appendix which generally does not need to be delivered each time it is updated. The activities, products, computer software configuration items (CSCIs) or subsystems, and the tailoring account for the **What** and should be about 15% of the document. The types of facilities needed, including the development and test environments account for 10% of the document and answer **Where**. The schedule, process model, and milestones are part of **When** and account for 10%. Compliance with the Statement of Work (SOW) and company standards should be included in answering **Why** and account for 5%. Lastly, the process model, methodology, company standards and tools account for the remaining 50% and answer **How**. You will note that some topics are covered in more than one area as the perspective may be different.

The project planning document or SDP is often arranged as shown in the chart. These topics will resurface in various modules of this course.

How the SDP fits in the hierarchy of other planning documents

The SDP fits within the program framework. The Program Management and System Engineering planning typically precede the Software Engineering planning. For the SDP to be consistent with the program and system engineering planning activities, the software engineering organization must participate in the program and system planning. Also the SDP often drives other planning documents at lower levels, including the Configuration Management Plan and the Quality Assurance Plan. These documents generally cover topics including: engineering and manufacturing, test and evaluation, planning, risk management, logistics, configuration management, data management, support to modern concepts, tailoring, human factors, process maturity, concurrency, process improvement, metrics, and process models. The hierarchy of planning documents is as shown:

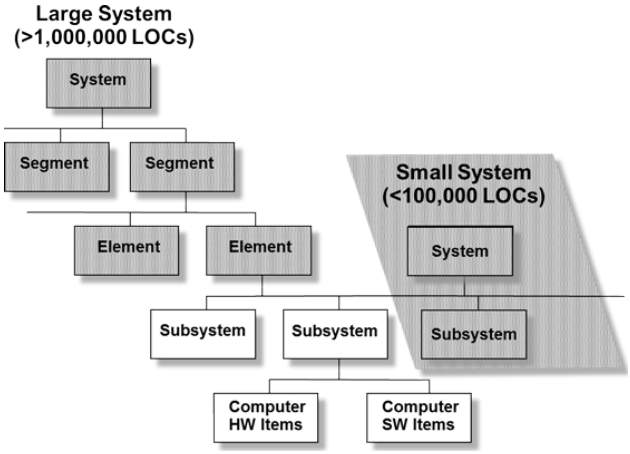


Software as a component of the system

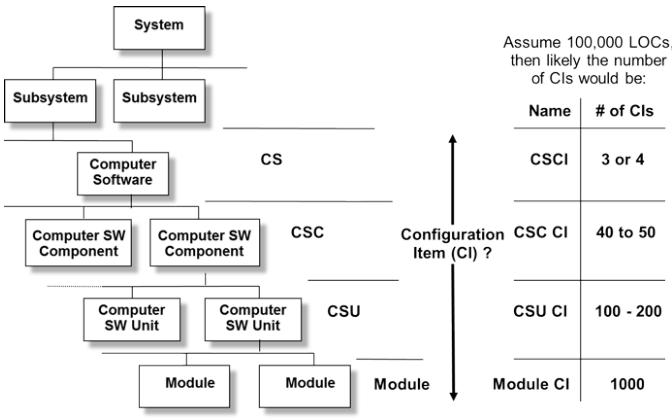
The Software Development Plan describes the planning for the development of the software components in the system; there is one SDP for the entire project. The other development documents including specifications, test procedures, and user manuals are produced for each software component designated as a "configuration item."

What is a Configuration Item?

No matter if the system is large or small, software is a component of the system. Systems are hierarchical.



A configuration item is a basic unit of a system and it can be as large or as small as necessary. The system is made up of subsystems, components, units and at the lowest level, modules. Generally there are 1000s of module configuration items. Groups of modules make up computer software unit configuration items which in turn make up computer software component configuration items which in turn make up the computer software configuration items which make up the system.



Computer Software (CS) is defined during requirements analysis; it satisfies an end-use function and is typically designated as a configuration item (CI). Each Computer Software Configuration Item (CSCI) requires its own set of specifications and technical reviews. For this reason the number of CSCIs should be minimal. Start by assuming there is one CSCI for the entire software system. Decompose it into CSCIs if:

- It crosses computer boundaries
- It crosses vendor boundaries including subcontractors and applications versus COTS
- One CSCI is too large; 30,000 LOCs is a guide
- Separate schedules are required
- Software requirements analysis results in the definition of more than one CSCI, that is, there are clearly very unique functions

Computer Software Component (CSC) is a functionally or logically distinct part of a CSCI. CSCs are identified during the preliminary design process.

Computer Software Unit (CSU) is a functionally or logically distinct part of a CSC and has the following characteristics:

- The unit performs a well-defined function.
- It is amenable to development by one person within the assigned schedule.
- Requirements can be traced to a unit.
- Implements a testable aspect of the requirements.
- It has a cyclomatic complexity of 10 or less.
- Typical units are between 100 and 1000 LOCs, but remember that each unit requires an SDF so larger units minimize SDF maintenance.
- It is composed of one or more modules.

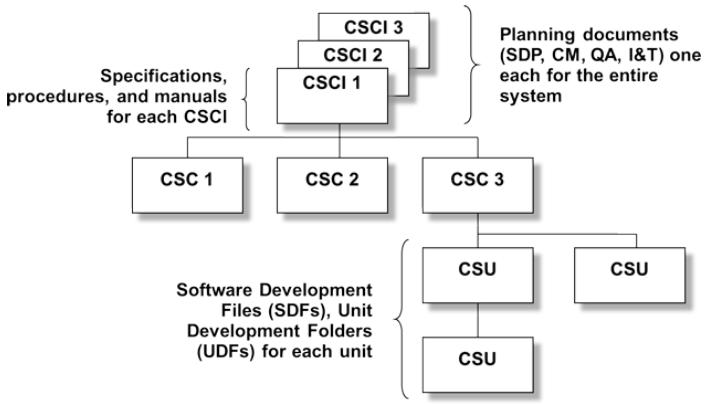
Module, the lowest component of software, is typically 100 LOCs or less.

The software developer keeps Software Development Files (SDFs) or Unit Development Files (UDFs) for each unit or folder.

The SDFs may include:

- Assumptions and constraints
- Design considerations
- Design documentation and data
- Source code items

- Unit and integrated unit test descriptions
- Unit and integrated unit test results
- Code evaluation, walkthrough, and inspection results
- Order of integration of the components
- Tests
- Any other valuable information about the software element

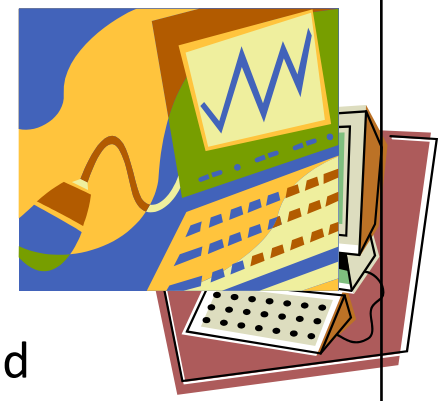


Software Estimation Methods

Software Estimation Methods

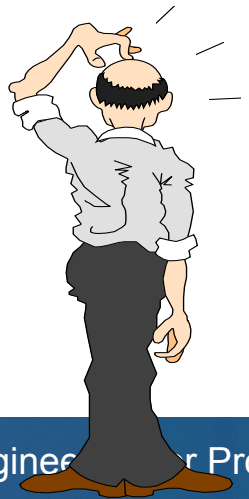
Computer Models

- Theory Based
- Empirical
- Regression Based

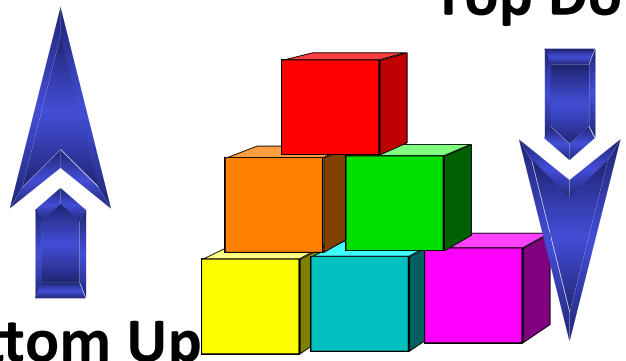


Price to Win

Top Down



Expert Judgment (Delphi)



Bottom Up

Software Estimation Methods

Estimation Method	Description	Advantages	Disadvantages
Regression-Based Models Examples: COCOMO, REVIC	Historical information is used to develop algorithms which relate cost with one or more software metrics producing a scatter diagram	• Objective, repeatable, easy to use • Can be calibrated to company or project environment	• Inputs may be subjective (need Lines of Code (LOCs)) • May not handle exceptional circumstances • May be based on inefficient past practices
Empirical-Based Models Example: Checkpoint	Estimate is based on analogy with cost of previously completed projects in the same domain	• Based on actual experience • Can break down cost at detailed level	• Not always clear how to compare projects • May miss differences between project applications and environments
Theory-Based Models Examples: SLIM, Price-S	Estimate is based on underlying theoretical considerations for software development processes	• Repeatable • Get what you pay for • Lots of research	• Proprietary • Expensive
Expert Judgment (Delphi)	Uses one or more experts to arrive at consensus estimate	• Can factor in differences between past projects and this project • Can factor in exceptions	• Only as good as the experts • Not repeatable
Price-to-Win Estimate	Estimate is based on whatever the customer has to spend	• Often wins the contract	• Schedule and budget are unrealistic • Engineers become demoralized
Top Down Estimate	Derive estimate from global properties of the product divided among components	• System level focus will not leave out system level functions	• Does not identify low level technical issues • Sometimes misses detailed components • Does not provide details for cost analysis
Bottom Up Estimate	Cost of each component is estimated, then costs are added for overall estimation	• Estimate for each component based on detailed understanding • Estimate backed up by personal commitment of individuals • Estimation errors balance out	• Can underestimate by overlooking system level costs • Requires more effort • Some cost elements may be included more than once

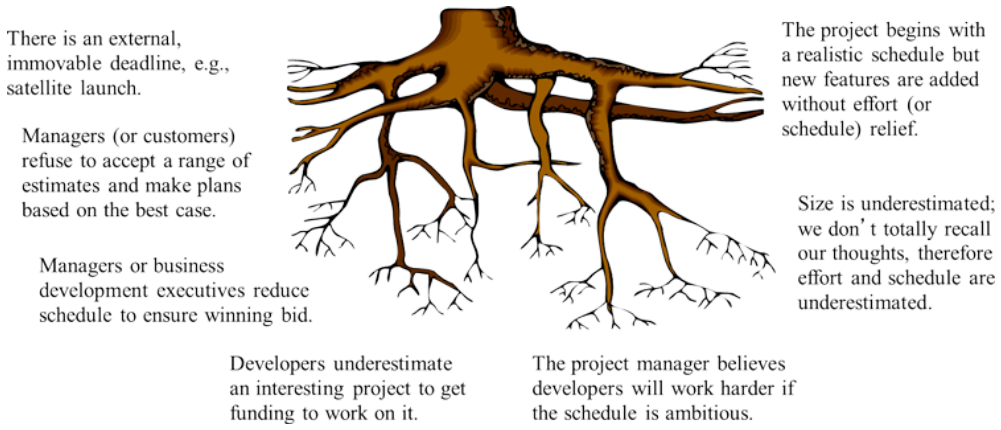
Scheduling

Cost and schedule estimation performed during proposal activity defines the initial software development schedule. This project plan or SDP must include a detailed software development schedule consistent with the program schedule. But beware...

> "Most IS people I know —managers or not— don't control their own schedules. Schedules are handed down, like stone tablets (or, some would say, like bat guano) from on high — where 'on high' could mean the marketing department or top management." — *Robert L. Glass, Software Practitioner, 1994*

> "Excessive or irrational schedules are probably the single most destructive influence in all of software." — *Capers Jones, 1994*

Software schedules are often overly optimistic. Some of the root causes are:



Scheduling Techniques

A variety of scheduling techniques are available, each supported by automated tools. Common scheduling tools include:

- Milestone Chart
- Gantt Chart, most common technique
- Critical Path Method (CPM) or Program Evaluation and Review Technique (PERT), both precedence network techniques

Milestone chart is the simplest of the scheduling techniques. It shows the due dates for the activities and these easy to manually change. It is difficult to see the relative time spans for each task or activity and it is hard to see the overlap in the activities. It is most suited for small projects. In this case, the team has completed the Preliminary Design Review (PDR) for Release 1. You can also see that while the Release 1 CSCI Integration and Test is due in November, the Software Requirements Review is due in December. Generally the next release can begin as soon as the previous release has begun coding, that is after the Critical Design Review (CDR). You will also note that the System Requirements Review and the System Design Review are not needed after Release 1. Typically the system requirements and design are defined in the first release and will not be changed in subsequent releases like the software details.

Milestone Chart

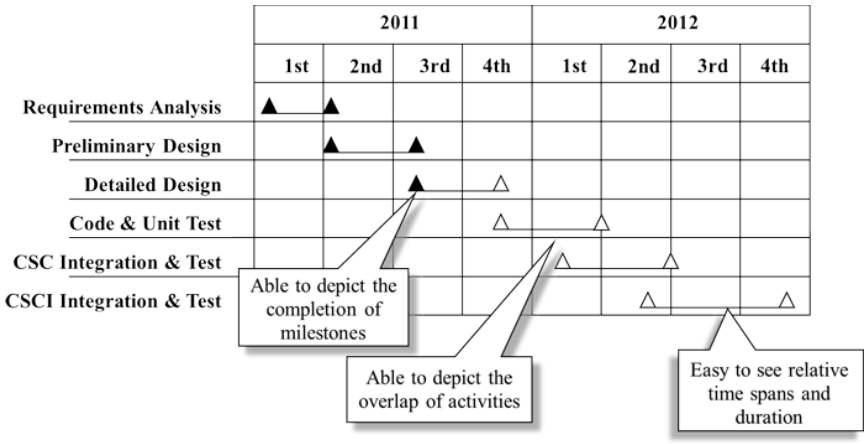
	System Requirements Review (SRR)	System Design Review (SDR)	Software Requirements Review (SWRR)	Preliminary Design Review (PDR)	Critical Design Review (CDR)	CSCI Integration and Test (I&T)	System Test
Release 1	**03/15/2010**	**06/15/2010**	**09/15/2010**	**01/15/2011**	06/01/2011	11/01/2011	01/01/2012
Release 2	NA	NA	12/01/2011	03/01/2012	08/01/2012	01/01/2013	03/01/2013
Release 3	NA	NA	01/01/2013	03/01/2013	07/01/2013	09/01/2013	11/01/2013

Difficult to see overlap in activities

Difficult to see relative time spans

Gantt chart is the most common technique, partially because it is used as part of the popular product, Microsoft Project. It provides a timeline across the top and the activities can be listed in hierarchical order on the left hand side. It shows the relative time spans and durations for each task and the overlap in the activities. This method also allows you to quickly identify if a task has started, in progress, or complete, by the shading of the triangle. A black triangle means that part of the task is complete. Thus in this example, the project has completed the Preliminary Design and Detailed Design has begun but is not yet complete.

Gantt Chart

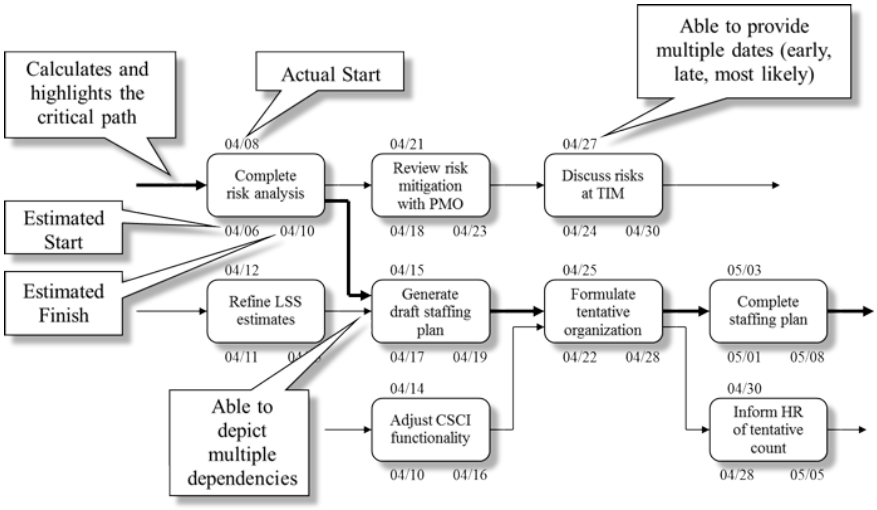


Critical Path Method (CPM) and Program Evaluation and Review Technique (PERT) are both precedence networks. Originally they were not the same, but today CPM and PERT are essentially the same. CPM was created in the late 1950s by Morgan R. Walker of DuPont and James E. Kelley, Jr. of Remington Rand. About the same time, Booz Allen Hamilton in conjunction with the US Navy developed PERT. CPM and PERT require a schedule to be developed that shows all activities, their durations, and the relationships between the activities, that is the relationship when a task ends and the subsequent task begins. It is important to note that CPM can have two meanings:

1. The **longest path through the network**, that is, the longest time to complete the entire project from the earliest start date to the last finish date so that any activity expansion or delay will extend the project time or
2. The more common definition that is the **minimum project time**, that is the least amount of total float such that any activity expansion or delay lengthens the project duration

For each activity, you determine the earliest start time, the earliest finish time, the latest start time and the latest finish time. You must perform a forward pass beginning with the earliest start date to find the earliest end date and then doing a backwards pass starting with the latest due date to determine the latest start date. **Float** or **slack** is the amount of time that an activity in a network can be delayed without causing a delay to subsequent tasks (free float) or the project completion date (total float). The critical path is the resulting path with the least amount of float or slack time allowing you to complete the activities before the project due date. The shortest time through the network is the most **optimistic time**, the longest time is most **pessimistic time**, and the time with the highest probability is the **most likely time**.

Precedence Networks:
Critical Path Network (CPM) or
Program Evaluation and Review Technique (PERT)



Here is a comparison of the scheduling techniques:

	Milestone Chart	**Gantt Chart**	**CPM**	**PERT**
	Generally used on small projects, no activity relationships	Most widely used, no activity relationships, depicts overlapping activities	Show task dependencies & provide means to identify the critical path	Show task dependencies & provide means to identify the critical path
Characteristics				
Complexity	Simplest	Simple	Complex	Complex
Suitable for large projects	Poor	Poor	Excellent	Excellent
Degree of control	Very low	Low	High	Highest
Easy to learn	Best	Excellent	Fair	Poor
Ease of manual calculations	Easiest	Easy	Hard	Hardest
Cost to prepare/maintain	Lowest	Low	High	Highest
Project scope	Poorest	Poor	Good	Excellent
Critical completion date	Fair	Fair	Good	Excellent
Frequent updating required	Easiest	Easy	Hard	Harder
Accuracy of projections	Fair	Fair	High	Highest
Appeal to Client	Good	Good	Excellent	Excellent

When and How to Develop the Software Development Plan (SDP)

The SDP is a comprehensive planning document that the software manager uses to direct the software development and/or maintenance, in other words, a plan to develop the software. It is often due with the submission of the proposal, but can be updated throughout the project. If not required with the proposal, it should be developed as early as possible subject to the completion of all planning activities including:

- Size, cost, and schedule estimation which discusses staffing, milestones, and schedule
- Selecting a process paradigm or process model
- Identifying the organization
- Tailoring decisions

The SDP or Project Plan is a team effort and has inputs, processes, and outputs. The inputs come from both internal and external sources. The project plans should address Capability Maturity Model Integration® (CMMI) requirements. Project Planning (PA):

- Is a level 2 (repeatable) Process Area (PA)
- Defines the purpose of software project planning to establish reasonable plans for performing the software engineering and managing the software project
- Describes the software project planning as:
    - Developing the estimates to do the work
    - Identifying and assessing software risks
    - Selecting the life cycle model
    - Establishing necessary commitments
    - Defining the plan to do the work

- **Internal Inputs**
    - Winning cost / schedule strategy
    - System Engineering Management Plan (SEMP)
    - Existing system capabilities (reuse)
    - Best practices & procedures
    - Work Breakdown Structure (WBS)
    - Design-to-cost strategies
- **Process**
    - Define software planning tasks
    - Identify resource sources
    - Refine software schedule
    - Generate software plan
    - Tailor software standards
    - Generate draft and final SDP
- **Outputs**
    - SDP
    - Subtasks
    - Task interdependencies
    - Detailed schedule
    - Tailored Standards
    - Rationale notes
    - Personnel requisitions
- **External Inputs (Acquisition Organization)**
    - Request for Proposal (RFP)
    - Statement of Work (SOW)
    - Contract Data Requirements List (CDRL)
- **External Inputs (Customer Requirements)**
    - Environment
    - Mission requirements
    - Operational scenarios
    - Operability scenarios
    - Manning constraints
    - Specified operator tasks

The SDP generally covers the topics listed below. Along the right side are the topics that we cover in the course.

Software Development Plan

Section	Topic
1	Scope
1.1	Identification
1.2	System Overview
1.3	Document Overview

1.4……Relationship to Other Plans ← Planning
2……Referenced Documents
3……Overview of Required Work ← Must Be Compliant with Statement of Work
4……Plans for Performing General Software Development Activities
4.1……Software Development Process ← Development Process
4.2……General Plans for Software Development
4.2.1……Software Development Methods ← Development Methods
4.2.2……Standards for Software Products ← IEEE/EIA 12207, Corporate Standards, etc.
4.2.3……Reusable Software Products
4.2.4……Handling Critical Requirements
4.2.5……Computer Hardware Resource Utilization
4.2.6……Recording Rationale
4.2.7……Access for Acquirer Review ← Joint Application Development (JADS), SharePoint, Internet, etc.
5……Planning for Performing Detailed Software Development Activities
5.1……Project Planning and Oversight
5.2……Establishing a Software Development Environment
5.3 – 5.12……Development Life Cycle Activities ← Details to Project Notebook
5.13……Preparing for Software Transition
5.14……Software Configuration Management ← CMMI ® Level 2 Process Area
5.15……Software Product Evaluation ← Formal Testing
5.16……Software Quality Assurance ← CMMI ® Level 2 Process Area
5.17……Corrective Action
5.18……Joint Technical Management Reviews
5.19……Other Software Development Activities
5.19.1……Risk Management ← Risk
5.19.2……Software Management Indicators ← Metrics, Software and Cost Management
5.19.3……Security and Privacy
5.19.4……Subcontract Management ← CMMI ® Level 2 Process Area
5.19.5……Interface with Software Independent Verification and Validation Agents ← Typically when multiply contractors are involved
5.19.6……Coordination with Associate Developers
5.19.7……Improvement of Project Processes ← Continuous Process Improvement
5.19.8……Other Activities Not Covered ← Training, Software Engineering Process Group
6……Schedule and Activity Network ← Planning, Scheduling
7……Project Organization and Resources ← Organization Structure, Staffing
7.1……Project Organization ← Organization Structure
7.2……Project Resources ← Staffing, Cost and Schedule Management
8……Notes and Appendices

Project Notebook

The Project Notebook (or a Standards and Procedures Manual) supplements the SDP. It:

- Establishes the detailed standards, procedures, guidelines, and restrictions that developers will follow to develop the software.
- Is produced incrementally or "just in time" for each phase of the development effort.
- Generally includes or references organizational standards.
- Allows the software manager to defer detailed instructions including naming conventions, tool usage conventions, and inspection checklists, until enough data is available to make meaningful decisions.
- Eliminates details from the SDP (Typically the SDP is a deliverable and subject to customer review and approval. With the details omitted, the customer reviews and approves the overall process but not the details such as the checklists in the Project Notebook.).

- Results in a SDP that does not overwhelm the developers. (The SDP may reach 75 pages, while the Project Notebook which is intended to be a referenced not read could be 300 pages.)

PROJECT NOTEBOOK

Section	Topic
1.....	Scope
2.....	Referenced Documents
3.....	Requirements Analysis
4.....	Preliminary Design
5.....	Detailed Design
6.....	Database Definition & Control
7.....	Code & Unit Test
8.....	Inspections
9.....	Program Support Library
Appendix A...	Directives
Appendix B...	Project History