SOFTWARE ENGINEERING (CAS725)

Dr. Ghanshyam S. Bopche
Assistant Professor
Dept. of Computer Applications

September 24, 2021

SOFTWARE ENGINEERING (CAS725) **Syllabus**

- **Unit-I**: Introductory concepts, The evolving role of software, Its characteristics, components and applications, A layered technology, The software process, Software process models, Software Development Life cycle, Software process and project metrics, Measures, Metrics and Indicators, Ethics for software engineers.
- **Unit-II**: Software Project Planning, Project planning objectives, Project estimation, Decomposition techniques, Empirical estimation models, System Engineering, Risk management, Software contract management, Procurement Management.
- **Unit-III**: Analysis and Design, Design concept and Principles, Methods for traditional, Real time of object oriented systems, Comparisons, Metrics, Quality assurance

- **Unit-IV**: Testing fundamentals, Test case design, White box testing, Basis path testing, Control structure testing, Black box testing, Strategies: Unit testing, integration testing, Validation Testing, System testing, Art of debugging, Metrics, Testing tools
- **Unit-V**: Formal Methods, Clean-room Software Engineering, Software reuse, Re-engineering, Reverse Engineering, Standards for industry.
- **References**:

1. Rajib Mall, "Fundamentals of Software Engineering", 4th Edition, PHI, 2014.
2. Roger S. Pressman, "Software Engineering-A practitioner's approach", 7 th Edition, McGraw Hill, 2010.
3. Ian Sommerville, "Software engineering", 10th Edition, Pearson education Asia, 2016.
4. Pankaj Jalote, "An Integrated Approach to Software Engineering", Springer Verlag, 1997.

5. James F Peters, Witold Pedrycz, “Software Engineering – An Engineering Approach”, John Wiley and Sons, 2000.
6. Ali Behforooz, Frederick J Hudson, “Software Engineering Fundamentals”, Oxford University Press, 2009.
7. Bob Emery , “Fundamentals of Contract and Commercial Management”,Van Haren Publishing, Zaltbommel, 2013

System Engineering

Goal: deciding what requirements should be implemented in hardware and what in software.

- System engineering is the activity of designing entire system, taking into account the characteristics of hardware, software, and human elements of these systems.
- Systems are developed to support human activities - work, entertainment, communication, protection of people and the environment, and so on.
- Systems interact with people, and their design is influenced by human and organizational concerns.

System Engineering (cont.)

- Systems that include software fall into two categories:
  1. **Technical computer-based systems**
     - Systems that include hardware and software components but not procedures and processes (e.g., televisions, mobile phones, and other equipment with embedded software).
     - Individual and organizations use technical systems for a particular purpose, but the knowledge of this purpose is not part of the technical systems.
  2. **Sociotechnical systems**
     - Include one or more technical systems but, crucially, also people, who understands the purpose of the system, within the system itself.
     - Defines operational processes, and people (the operators) are inherent part of the system.
     - Governed by organizational policies and rules and may be affected by external constraints such as national laws and regulatory policies.

System Engineering (cont.)

- **System engineering** includes procurement, specification, development, deployment, operation, and maintenance of both technical and sociotechnical systems.
- **System engineers**
  - have to consider the capabilities of hardware and software as well as system's interactions with users and its environment.
  - must think about the system's services, the constraint under which the system must be build and operated, and the ways in which the system must be used.

System Engineering (cont.)

Stages in the lifetime of large, complex systems

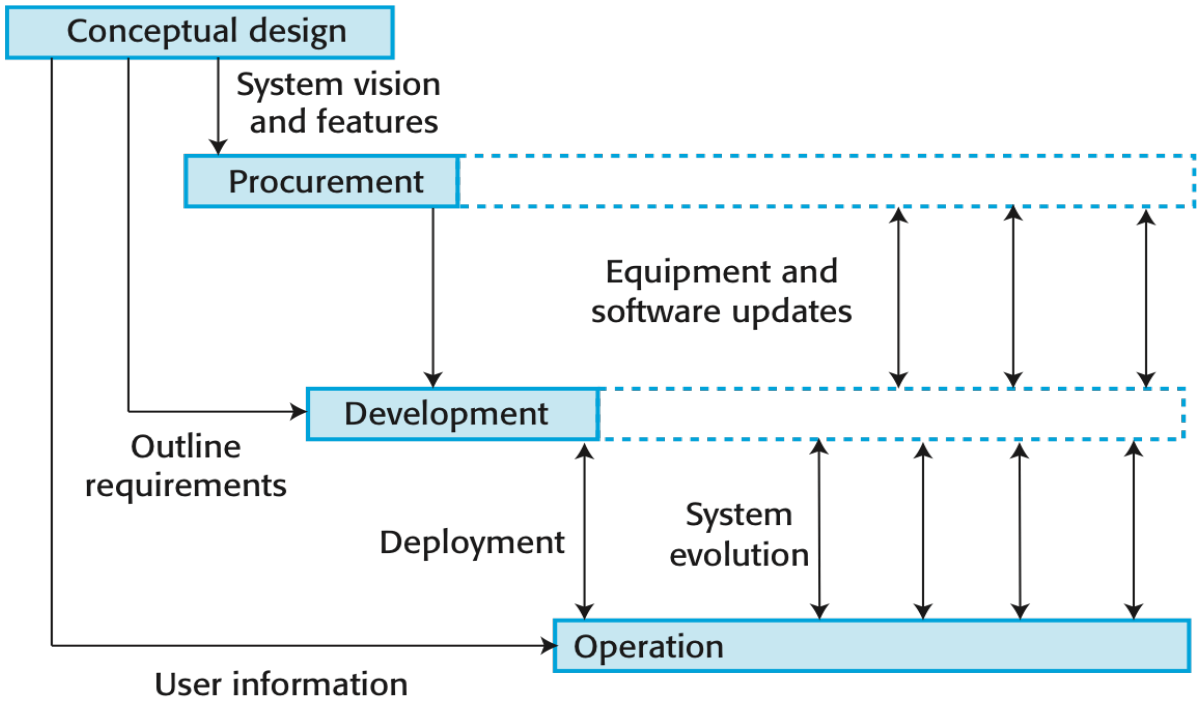


Figure: Stages of systems engineering

System Engineering (cont.)

Involvement of professionals in the development of large, complex system

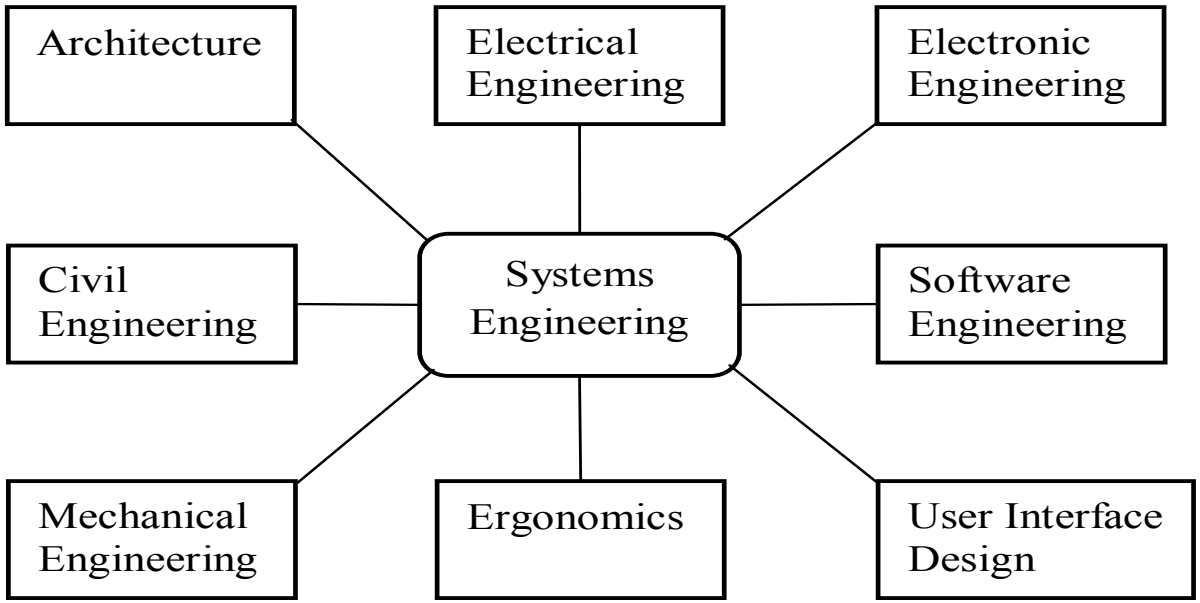


Figure: Professional disciplines involved in ATC system engineering

System Engineering (cont.)

1. Conceptual Design

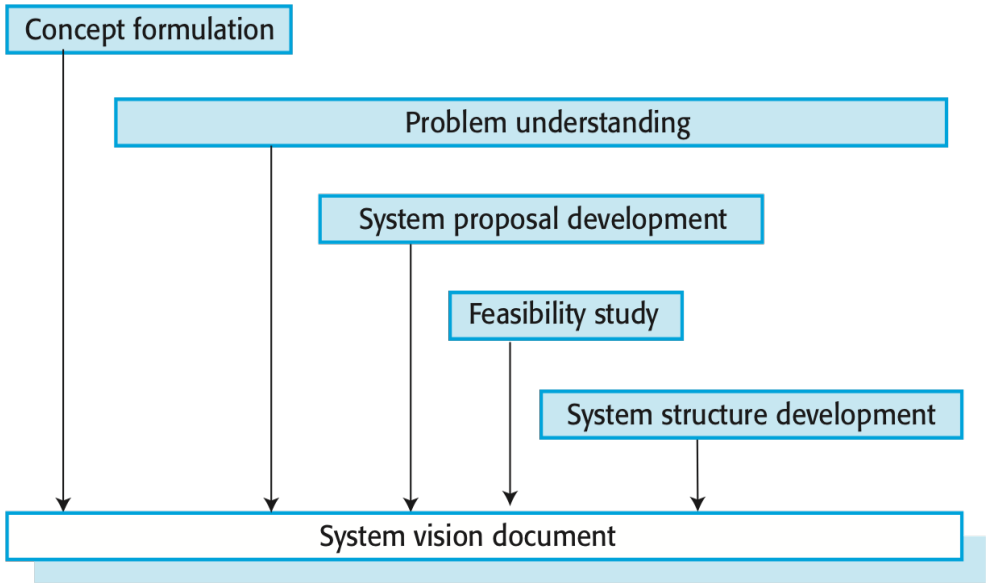


Figure: Conceptual design activities

System Engineering (cont.)

2. **System Procurement**
   - A process whose outcome is a decision to buy one or more systems from system suppliers.
   - What type of decisions to be made?
     - Scope of the system that is to be purchased,
     - System budgets and timescales,
     - High-level system requirements.
   - Based on the above information, further decisions are then made on whether to procure a system, the type of system required, and the supplier or suppliers of the system.

System Engineering (cont.)

- What drives procurement decisions?
  - The replacement of other organizational systems
  - The need to comply with external regulations
  - External competition
  - Business reorganization
  - Available Budget

System Engineering (cont.)

- What type of systems or system components may have to be procured?
  - **Off-the-shelf applications**: applications that may be used without change or that need only minimal configuration for use.
  - **Configurable applications**: applications that have to be modified or adapted for use either by modifying the code or by using inbuilt configuration features, such as process definition and rules.
  - **Custom systems** that have to be specially designed and implemented for use.

System Engineering (cont.)

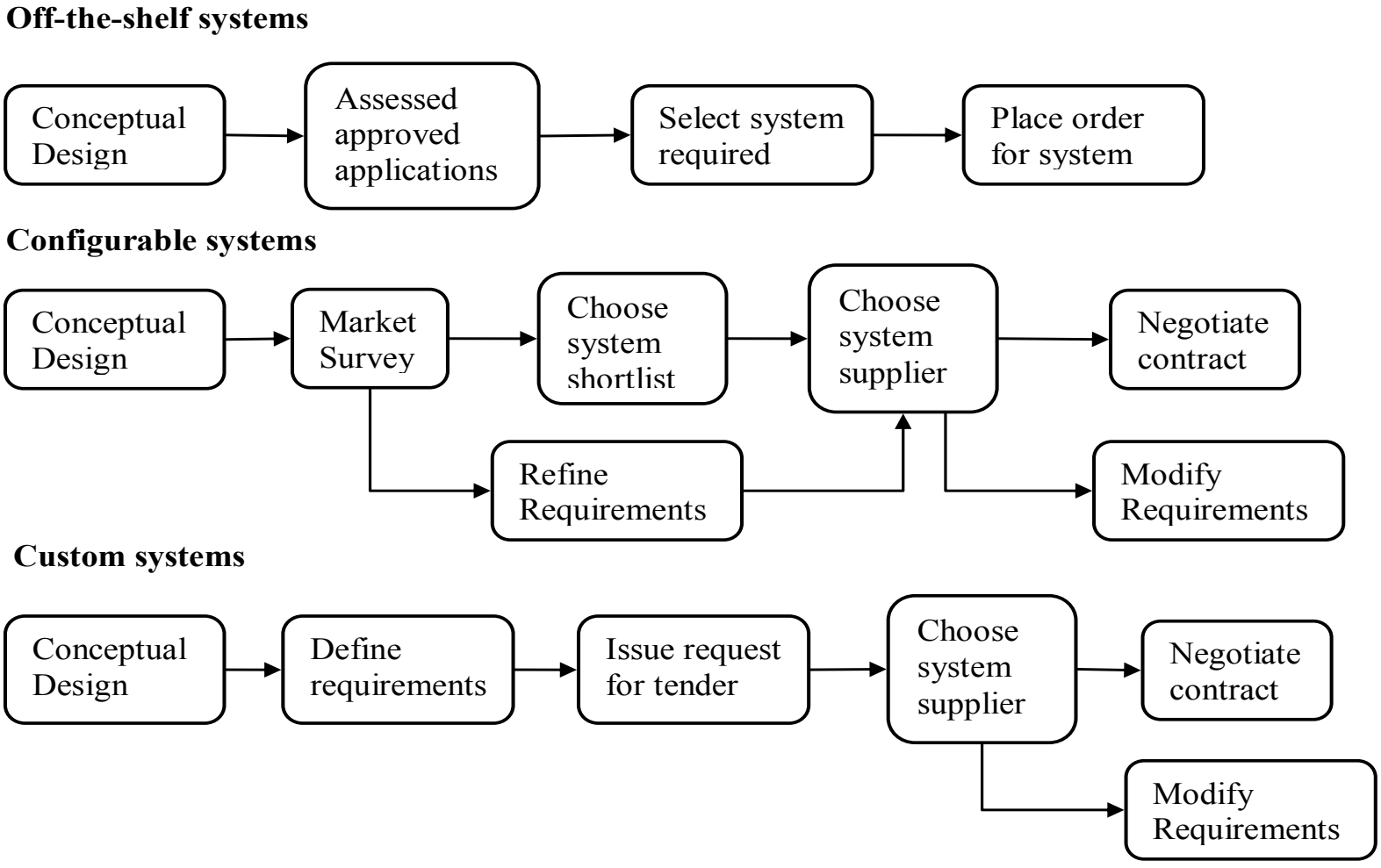


Figure: System procurement processes

System Engineering (cont.)

3. System Development

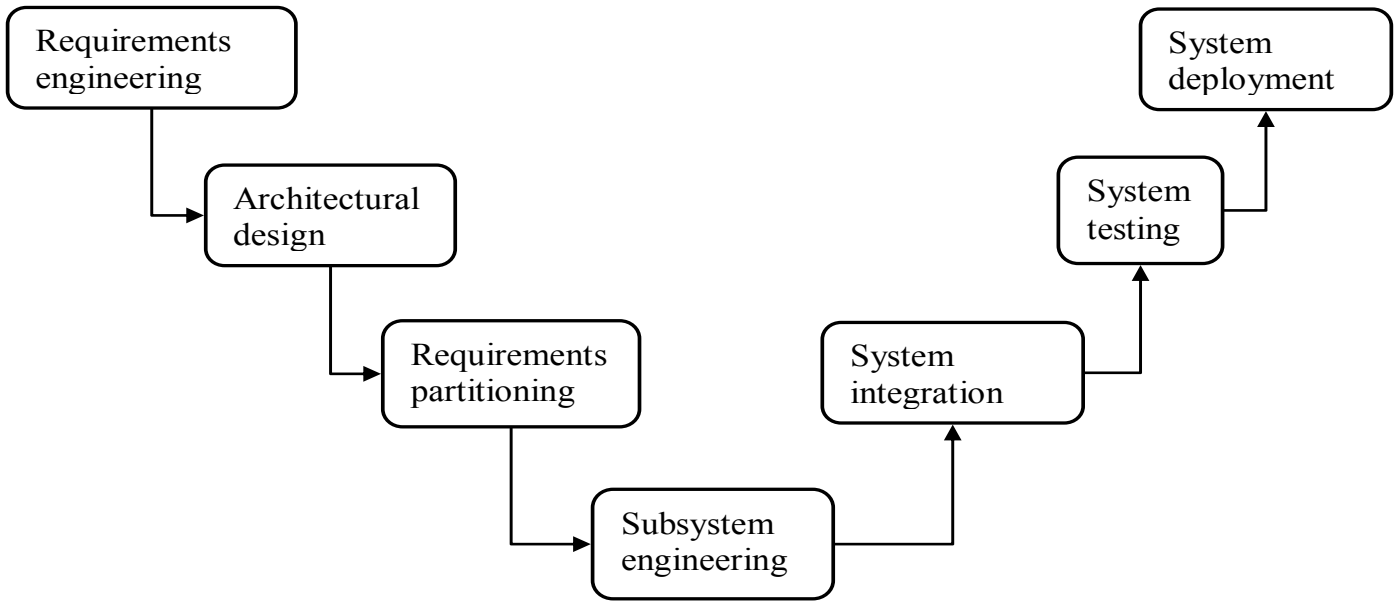


Figure: The system development process

System Engineering (cont.)

4. **System operation and Evolution**
   - **System Operation**
     - The processes that are involved in using the system as intended by its designers.
     - For new system, the operational processes need to be defined and documented during the system development process.
     - Operators may have to be trained and other work processes adapted to make effective use of the new system.
     - The operational processes to be flexible and adaptable.

System Engineering (cont.)

- **System Evolution**
  - A process that runs alongside normal system operational processes.
  - Involves reentering the development process to make changes and extensions to the system's hardware, software, and operational processes.
  - System evolution is costly.

Risk Management

What is Risk?

- A potential problem that might occur.
- Risk in the context of Software Engineering include following concern:
  - **The future**
    - What risks might cause the software to project to go awry?
  - **The change**
    - How will changes in customer requirements, development technologies, target environments, and all the entities connected to the project affect timeliness and overall success?
  - **The choices**
    - What methods and tools should be used?
    - How many peoples should be involved?
    - How much emphasis on quality is enough?

Reactive versus Proactive Risk Strategies

- **Reactive Risk Strategies**
  - "Never worrying about the problems until they happened."
  - Monitoring of software projects for likely risks.
  - Resources are set aside to deal with risks.
  - Software team does nothing about risks until something goes wrong.
  - The team flies into action in an attempt to correct the problem (fire-fighting mode).

Reactive versus Proactive Risk Strategies (cont.)

- **Proactive Risk Strategies**
  - Primary objective is to avoid risks (begins long before technical work is initiated).
  - **Steps**
    - Potential risks are identified,
    - Their probability and impact are assessed,
    - Risks are ranked by their importance,
    - Team establishes a plan for managing risks (a contingency plan that will enable team to respond in a controlled and effective manner).

Software Risks

1. **Project Risks**
   - Threatens project plan.
   - Project schedule will slip and the development cost will increase.
   - Project risks identify potential budgetary, schedule, personnel (staffing and organization), resource, stakeholder, and requirements problems and their impact on software project.
   - Project estimation risk factors: project complexity, size, the degree of structural uncertainty.

Software Risks (cont.)

2. **Technical Risks**
   - Threaten the quality and timeliness of the software to be produced.
   - Occur because of the problem is harder to solve than it thought to be.
   - Because of technical risk, project implementation may become difficult or impossible.
   - Technical risks identify potential design, implementation, interface, verification, and maintenance problems.
   - Additional technical risk factors: specification ambiguity, technical uncertainty, technical obsolescence, and "leading-edge" technology.

Software Risks (cont.)

3. **Business Risks**
   - Threaten the viability of the software to be built, and often jeopardize the project or the product.
   - Potential business risks
     - 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).

Categories of Risk

- **Generic risks**
  - potential threat to every software project.

.

- **Product specific risks**
  - 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.
  - To identify product-specific risks, the project plan and the software statement of scope are examined.
  - Question to be answered:
    "What special characteristics of this product may threaten our project plan?"

Risk Identification

- Systematic approach to identify threats to the project plan (estimates, schedule, resource loading, etc.)
- **Risk Checklist**: method of identifying risks
- Checklist focuses on 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 of 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.

Risk Identification (cont.)

- **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.

Assessing overall project risk

Questions that need to be answered for each software project (derived from risk data obtained by surveying experienced software project managers).

- Have to software and customer managers formally committed to support the project?
- Are end users enthusiastically committed to the project and the system/product to be built?
- Are requirements fully understood by the software engineering team and its customers?
- Have customers been involved fully in the definition of requirements?
- Do end users have realistic expectations?
- Is the project scope stable?
- Does the software engineering team have the right mix of skills?

Assessing overall project risk (cont.)

- Are project requirements stable?
- Does the project team have experience with the technology to be implemented?
- Is the number of people on the project team adequate to do the job?
- Do all customer/user constituencies agree on the importance of the project and on the requirements for the system/product to be built?

Note: the degree to which the project is at risk is directly proportional to the number of negative responses to these questions.

Risk Components and Drivers

The project manager must identify the risk drivers that affect software risk components - performance, cost, support, and schedule.

- **Performance risk**: the degree of uncertainty that the product will meet its requirements and be fit for its intended use.
- **Cost risk**: the degree of uncertainty that the project budget will be maintained.
- **Support risk**: the degree of uncertainty that the resultant software will be easy to correct, adapt, and enhance.
- **Schedule risk**: the degree of uncertainty that the project schedule will be maintained and that the product will be delivered on time.

Note: the impact of each risk driver on the risk component is divided into one of the four impact categories: negligible, marginal, critical, or catastrophic.

Risk Components and Drivers (cont.)

Impact Assessment

Category \ Components		Performance	Support	Cost	Schedule
Catastrophic	1	Failure to meet the requirement would result in mission failure		Failure results in increased costs and schedule delays with expected values in excess of $500K	
	2	Significant degradation to nonachievement of technical performance	Nonresponsive or unsupportable software	Significant financial shortages, budget overrun likely	Unachievable IOC
Critical	1	Failure to meet the requirement would degrade system performance to a point where mission success is questionable		Failure results in operational delays and/or increased costs with expected value of $100K to $500K	
	2	Some reduction in technical performance	Minor delays in software modifications	Some shortage of financial resources, possible overruns	Possible slippage in IOC
Marginal	1	Failure to meet the requirement would result in degradation of secondary mission		Costs, impacts, and/or recoverable schedule slips with expected value of $1K to $100K	
	2	Minimal to small reduction in technical performance	Responsive software support	Sufficient financial resources	Realistic, achievable schedule
Negligible	1	Failure to meet the requirement would create inconvenience or nonoperational impact		Error results in minor cost and/or schedule impact with expected value of less than $1K	
	2	No reduction in technical performance	Easily supportable software	Possible budget underrun	Early achievable IOC

Note: (1) The potential consequence of undetected software errors or faults.
(2) The potential consequence if the desired outcome is not achieved.

Risk Components and Drivers (cont.)

Figure: Impact assessment

Risk Projection

- Also called risk estimation.
- Attempts to rate each risk in two ways
  1. the likelihood or probability that the risk is real and
  2. the consequences of the problems associated with the risk.
- Risk projection steps
  - Establish a scale that reflects the perceived likelihood of a risk.
  - Delineate the consequence of the risk.
  - Establish 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 misunderstanding.

Risk Projection (cont.)

Risk Table: simple technique for risk projection.

Risks	Category	Probability	Impact	RMMM
Size estimate may be significantly low	PS	60%	2	
Larger number of users than planned	PS	30%	3	
Less reuse than planned	PS	70%	2	
End users resist system	BU	40%	3	
Delivery deadline will be tightened	BU	50%	2	
Funding will be lost	CU	40%	1	
Customer will change requirements	PS	80%	2	
Technology will not meet expectations	TE	30%	1	
Lack of training on tools	DE	80%	3	
Staff inexperienced	ST	30%	2	
Staff turnover will be high	ST	60%	2	
Σ				
Σ				
Σ				

Impact values:
1—catastrophic
2—critical
3—marginal
4—negligible

Figure: Sample risk table prior to sorting

Risk Projection (cont.)

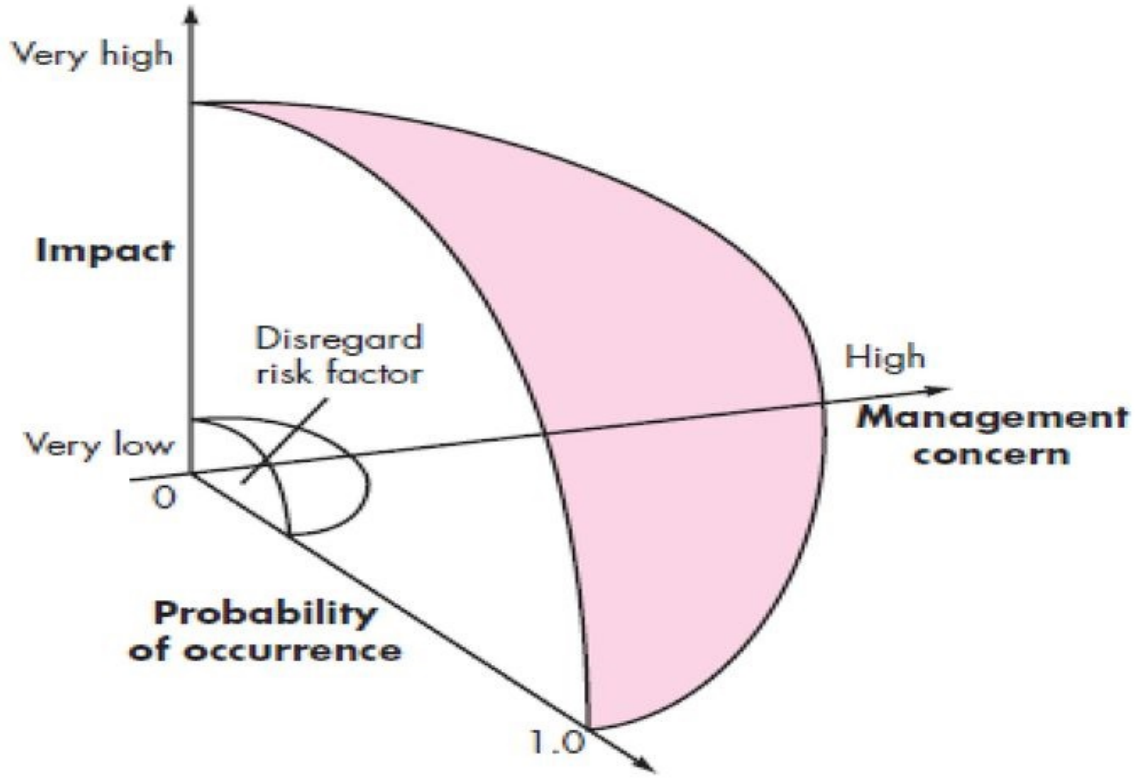


Figure: Risk and management concern

- Risk factor with high impact but a very low probability of occurrence should not absorb a significant amount of management time.
- High-impact risks with moderate to high probability and low-impact risks with high probability should be carried forward into the risk analysis step that follow.

Risk Projection (cont.)

Assessing risk impact

- Three factors affect the consequences that are likely if a risk does occur.
  - **The nature of the risk**: indicates the problems that are likely if it occur.
  - **The scope of a risk**: combines the severity (just how serious it is?) with its overall distribution (how much of the project will be affected or how many stakeholders are harmed?)
  - **Timing of a risk**: considers when and for how long the impact will be felt.
- The overall risk exposure (RE) can be computed as

$$RE = P \times E$$

where

- P $\rightarrow$ the probability of occurrence of a risk, and
- C $\rightarrow$ the cost to the project should the risk occur.

Risk Mitigation, Monitoring, and Management

- An effective strategy for dealing with risk must consider three issues:
  - risk avoidance,
  - risk monitoring, and
  - risk management and contingency planning
- **Risk avoidance**: proactive approach to risk, achieved by developing a plan for risk mitigation.
- **Risk monitoring**
  - The project manager monitors factors that may provide an indication of whether the risk is becoming more or less likely.
  - Additionally, the project manager should monitor the effectiveness of risk mitigation steps.
- **Risk management and contingency planning**: assumes that mitigation efforts have failed and that the risk has become a reality.

Risk Mitigation, Monitoring, and Management (cont.)

- **The RMMM Plan**
  - Document all work performed as part of risk analysis.
  - Used by the project manager as a part of the overall project plan.
- **Risk information sheet (RIS)**: each risk is documented individually.

Risk Mitigation, Monitoring, and Management

(cont.)

Risk information sheet

Risk ID: P02-4-32	Date: 5/9/02	Prob: 80%	Impact: high

Description:
Only 70 percent of the software components scheduled for reuse will, in fact, be integrated into the application. The remaining functionality will have to be custom developed.

Refinement/context:
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.

Mitigation/monitoring:
1. Contact third party to determine conformance with design standards.
2. Press for interface standards completion; consider component structure when deciding on interface protocol.
3. Check to determine number of components in subcondition 3 category; check to determine if language support can be acquired.

Management/contingency plan/trigger:
RE computed to be $20,200. Allocate this amount within project contingency cost. Develop revised schedule assuming that 18 additional components will have to be custom built; allocate staff accordingly.
Trigger: Mitigation steps unproductive as of 7/1/02

Current status:
5/12/02: Mitigation steps initiated.

Originator: D. Gagne	Assigned: B. Laster

Figure: Risk information sheet

SOFTWARE ENGINEERING (CAS725)

Dr. Ghanshyam S. Bopche
Assistant Professor
Dept. of Computer Applications

September 20, 2021

SOFTWARE ENGINEERING (CA725)

Syllabus

- **Unit-I**: Introductory concepts, The evolving role of software, Its characteristics, components and applications, A layered technology, The software process, Software process models, Software Development Life cycle, Software process and project metrics, Measures, Metrics and Indicators, Ethics for software engineers.
- **Unit-II**: Software Project Planning, Project planning objectives, Project estimation, Decomposition techniques, Empirical estimation models, System Engineering, Risk management, Software contract management, Procurement Management.
- **Unit-III**: Analysis and Design, Design concept and Principles, Methods for traditional, Real time of object oriented systems, Comparisons, Metrics, Quality assurance

SOFTWARE ENGINEERING (CA725) (cont.)

- **Unit-IV**: Testing fundamentals, Test case design, White box testing, Basis path testing, Control structure testing, Black box testing, Strategies: Unit testing, integration testing, Validation Testing, System testing, Art of debugging, Metrics, Testing tools
- **Unit-V**: Formal Methods, Clean-room Software Engineering, Software reuse, Re-engineering, Reverse Engineering, Standards for industry.
- **References**:

1. Rajib Mall, "Fundamentals of Software Engineering", 4th Edition, PHI, 2014.
2. Roger S. Pressman, "Software Engineering-A practitioner's approach", 7 th Edition, McGraw Hill, 2010.
3. Ian Sommerville, "Software engineering", 10th Edition, Pearson education Asia, 2016.

SOFTWARE ENGINEERING (CA725) (cont.)

4. Pankaj Jalote, "An Integrated Approach to Software Engineering", Springer Verlag, 1997.
5. James F Peters, Witold Pedrycz, "Software Engineering – An Engineering Approach", John Wiley and Sons, 2000.
6. Ali Behforooz, Frederick J Hudson, "Software Engineering Fundamentals", Oxford University Press, 2009.
7. Bob Emery , "Fundamentals of Contract and Commercial Management",Van Haren Publishing, Zaltbommel, 2013

The Project Planning Process

- **Objective**: provides a framework that enables the project manager to make reasonable estimates of resources, cost, and schedule.
  - Estimates define best-case and worst-case scenarios so that project outcomes can be bounded.

Task Set for Project Planning

1. Establish project scope
2. Determine feasibility
3. Analyze potential risks
4. Define required resources
   - 4.1 Determine required human resources
   - 4.2 Determine reusable software resources
   - 4.3 Identify environmental resources
5. Estimate cost and effort
   - 5.1 Decompose the problem
   - 5.2 Develop two or more estimates using size, function points, process tasks, or use cases.
   - 5.3 Reconcile the estimate.
6. Develop a project schedule
   - 6.1 Establish a meaningful task set
   - 6.2 Define a task network
   - 6.3 Use scheduling tools to develop a time-line chart
   - 6.4 Define schedule tracking mechanisms

Software Scope and Feasibility

Software Scope: describes

- the functions and features that are to be delivered to end users;
- the data that are input and output;
- the content that is presented to users as a consequence of using the software; and
- the performance (processing and response time consideration), constraints (limits placed on the software by external hardware, available memory, or other existing systems), interfaces, and reliability that bound the system.

Software Scope and Feasibility (cont.)

Software Scope is defined using one of the two techniques:

1. A narrative description of software scope is developed after communication with all stakeholders.
2. A set of use cases is developed by end users.

Dimensions of Software Feasibility

1. **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?
2. **Finance**
   - Is it financially feasible?
   - Can development be completed at a cost the software organization, its clients, or the market can afford?
3. **Time**
   - Will the project's time-to-market beat the competition?
4. **Resources**
   - Does the organization have the resources needed to succeed?

Software project planning activity

- If feasibility study is positive $\rightarrow$ go for software planning.
- Feasibility studies of a software:
  - Technical Feasibility $\rightarrow$ Resource Allocation (Software Organization)
  - Economic Feasibility $\rightarrow$ Profit Evaluation (Software Organization)
  - Operational Feasibility $\rightarrow$ Functional Feasibility (Client Organization)
- Software $\rightarrow$ Technically Feasible $\propto \frac{1}{\textit{Economically Feasible}}$

Software project planning activity (cont.)

- Software Project planning consists of following five major activities:
  - Estimation
  - Scheduling
  - Risk Analysis
  - Quality management planning
  - Change management planning
- Essence of Software Planning Activities
  - Resource Estimation
  - Effort Estimation
  - Cost Estimation
  - Time Estimation

Estimation of the Resources

- Required to estimate the software development effort.
- Major categories of software engineering project resources : 1) People 2) Reusable software components, and 3) The development environment (hardware and software tools)

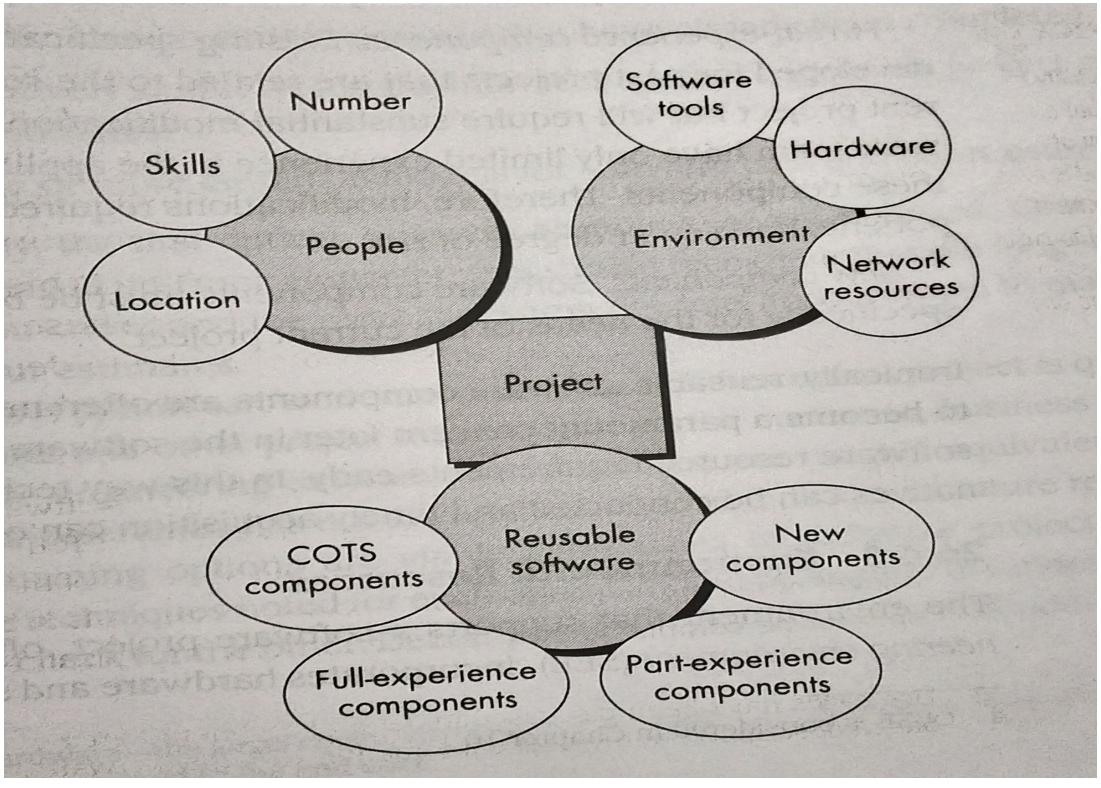


Figure: Software project resources

Estimation of the Resources (cont.)

1. **Human Resource**
   - Focuses on the human skills required to complete the software development.
   - The number of people required for a software project can be determined only after an estimate of development effort (e.g., person-months) is made.

Estimation of the Resources (cont.)

2. **Reusable software resources**
   - Focus is on the creation and reuse of software building blocks (components).
   - Such components must be cataloged for easy reference, standardized for easy application, and validated for easy integration.

   2.1 **Off-the-shelf components**
   - Existing software that can be acquired from a third party or from a past project.
   - COTS (commercial off-the-shelf) components are purchased from a third party, are ready for use on the current project, and have been fully validated.

Estimation of the Resources (cont.)

2.2 Full-experience components

- Existing specifications, designs, code, or test data developed for past projects that are similar to the software to built for the current project.
- Members of the current software team have had full experience in the application area represented by these components.
  → Therefore, modifications required for full-experience components will be relatively low risk.

2.3 Partial-experience components

- Similar to fully-experience components but will require substantial modification.
- Members of the current software team have only limited experience in the application area represented by these components.
  → Therefore, modifications required for partial-experience components have a fair degree of risk.

2.4 New components

- Must be built by the software team specifically for the needs of the current project.

Estimation of the Resources (cont.)

3. **Environmental software resources**
   - Software engineering environment (SEE): the environment that supports a software project (incorporates hardware and software).
   - Hardware: provides a platform that supports the tools (i.e., softwares) required to produce the work products (outcome of good software engineering practice).
   - When a computer-based system (incorporating specialized hardware and software) is to be engineered, the software team may require access to hardware elements being developed by other engineering teams.

Software Project Estimation

To achieve reliable cost and effort estimates, following techniques are available:

- **Delay Estimation** (until late in the project)
  - We can achieve 100% accurate estimates after the project is complete.
  - Not practical.
- **Base estimates** (on similar projects that have already been completed)
  - Works well only if the current project is quite similar to past efforts and other project influences (e.g., the customer, business conditions, the software engineering requirements, deadlines) are roughly equivalent.

Software Project Estimation (cont.)

- Use of relatively simple **decomposition techniques** to generate project cost and effort estimates.
  - Uses divide-and-conquer approach.
  - By decomposing the project into major functions and related software engineering activities, cost and effort estimation can be performed in a stepwise fashion.
- Use of one or more **empirical models** for software cost and effort estimation.
  - Complement to decomposition techniques.
  - Empirical models are based on historical data.
  - Historical data is used to seed the estimate.

Decomposition Techniques

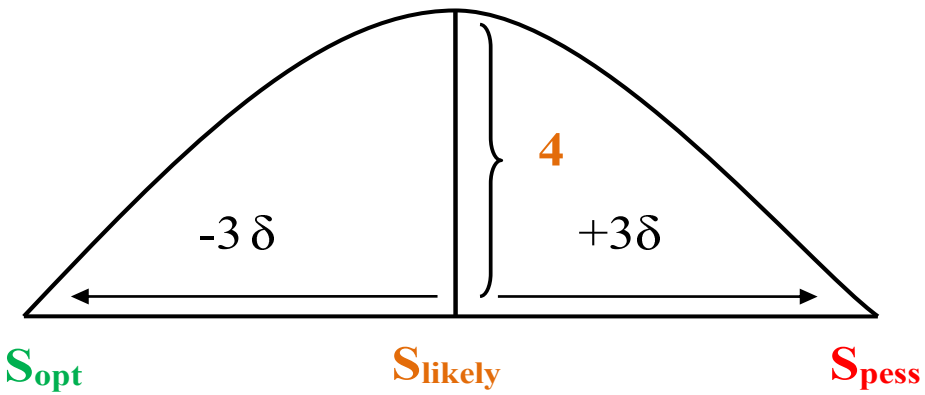
- **Motivation**: the problem to be solved (i.e., developing a cost and effort estimates for a software project) is too complex to be considered in one piece.
- **Solution**: decompose the problem, recharacterize it as a set of smaller, manageable problems.
- **Decomposition Techniques**
  1. **Problem-based decomposition**
     - 1.1 Direct Measure (size can be measured in lines of code (LOC))
     - 1.2 Indirect Measure (size is represented as function points (FPs))
  2. **Process-based decomposition**

Problem-based Estimation

- Putnam and Myers suggest that the results of each of the size-based approach be combined statically to compute a *three-point* or *expected-value* estimate.
  - This is accomplished by developing optimistic (low), most likely, and pessimistic (high) values for size and combining them using the equation.
- 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 using the following equation

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

Problem-based Estimation (cont.)



- S gives heaviest credence to the "most likely" estimate and follows a beta probability distribution.
- **Assumption**: there is a very small probability the actual size result will fall outside the optimistic or pessimistic values.

Problem-based Estimation (cont.)

(1) Direct Measure (LOC-based approach)

Case Study: Banking Software

- **Major software functions/modules**: Graphical User Interface (GUI), DB Access, Server Part, and Client Part.
- A range of LOC estimates is developed for each function. For example, the range of LOC estimates for the GUI functionality is optimistic, 4500 LOC; most likely, 5800 LOC; and pessimistic, 6900 LOC.
- Applying the equation
  $S_{GUI} = \frac{(S_{opt}+4\times S_m+S_{pess})}{6} = \frac{(4500+4\times 5800+6900)}{6} = \frac{34600}{6} = 5766$
  $\therefore$ The expected value for the GUI function is 5766 LOC.
- Other estimates are derived in a similar fashion.

Problem-based Estimation (cont.)

Function/Module	**Estimated LOC**
Graphical User Interface (GUI)	5766
DB Access	8000
Server Part	6500
Client Part	6000
Count total	**26266** LOC (Large Project)

- Organizational average productivity for system of this type $\rightarrow$ 850 LOC/PM (Based on review of historical data).
- Labor rate/salary $\rightarrow$ \$5000
- Effort $= \frac{\textit{Project Size in KLOC}}{\textit{Productivity}} = \frac{26266}{850} = 30.9 \equiv 31$ person-month
- Cost of developing a software $= \textit{Effort} \times \textit{Pay}$
  $= 31 \times 5000 = 155000 = \$155$ K
- Cost per Line of Code $= \frac{\textit{Project Cost}}{\textit{Total LOC}} = \frac{155000}{26266} = 5.9 \equiv \$6$

Problem-based Estimation (cont.)

LOC-based Basic Metrics

1. Effort $= \frac{Project\ Size\ (in\ LOC)}{Productivity}$
2. Productivity $= \frac{Project\ Size\ (in\ LOC)}{Effort}$
3. Cost of a Software $= Effort \times Pay$
4. Quality of a Software $= \frac{Errors}{KLOC}$
5. Cost per Line of Code $= \frac{Cost\ of\ a\ Software}{Project\ Size\ (in\ LOC)}$
6. Documentation = Pages per documentation (P_{pdoc}) for KLOC

Size-oriented Table

Project Title	Size	Cost	Effort	Errors	Defects	P_{pdoc}	People
SafeHome	36 KLOC	\$500K	90	80	78	150	15
SafeBank	26 KLOC	\$155K	31	-	-	-	-
EaseMyWork	20 KLOC	\$145K	31	-	-	-	-

Problem-based Estimation (cont.)

(2) Indirect Measure (FP-based estimation)

- Function point: atomic operation
- Decomposition for FP-based estimation focuses on information domain values rather than software functions.
- **Domain classification**
  1. Information Domain
     - Number of external inputs
     - Number of outputs
     - Number of external inquiries
     - Number of internal logical files
     - Number of external interface files
  2. Functional Domain
     - Number of transformations
  3. Behavior Domain
     - Number of transitions

Problem-based Estimation (cont.)

Information Domain Value	Opt.	Likely	Pess.	Estimated Value	Weight	FP Count
Number of External Inputs	24	27	31	$27.17 \equiv 27$	4	108
Number of Outputs	20	23	27	$23.17 \equiv 23$	5	115
Number of External Inquiries	16	19	21	$18.833 \equiv 19$	4	76
Number of Internal Logical Files	4	8	7	$7.17 \equiv 7$	10	70
Number of External Interface Files	2	4	6	4	7	28
Count total						397

Weight: Simple, Average, and Complex

- Unadjustable function points = count total = 397
- Adjustable function points = count total $\times$ Effort adjustment factor (EAF)
- Effort adjustment factor (EAF) $= 0.65 + 0.01 \times \sum fi$

Problem-based Estimation (cont.)

Complexity weighting factors

Sr. No	Factor	Value
1	Backup and recovery	4
2	Data communications	3
3	Distributed processing	5
4	Performance critical	4
5	Existing operating environment	3
6	Online data entry	4
7	Input transactions over multiple screens	5
8	Master files updated online	3
9	Information domain values complex	5
10	Internal processing complex	5
11	Code designed for reuse	4
12	Conversion/installation in design	4
13	Multiple installations	5
14	Application designed for change	5
		59 ($\leq$ 70)

Subject Assessment

0 $\rightarrow$ Not desirable, 1 $\rightarrow$ Least desirable, 5 $\rightarrow$ Most desirable

Problem-based Estimation (cont.)

- Effort adjustment factor (EAF) $= 0.65 + 0.01 \times 59$
  EAF $= 0.65 + 0.59 = 1.24$
  $\therefore$ Adjustable function points $= 397 \times 1.24 = 492.28$
- Organizational average productivity for systems (of this type) $= 16$ FP/PM
- Labor Rate/Salary $= \$5000$
- Effort $= \frac{\textit{Total number of adjustable function points}}{\textit{Productivity}}$
  Effort $= \frac{492.28}{16} = 30.7675 \equiv 31$ person-month
- Cost of a Software $= \textit{Effort} \times \textit{Pay} = 31 \times \$5000 = \$155K$

Problem-based Estimation (cont.)

Function point-based Metrics

1. Effort $= \frac{\textit{Total number of adjustable FPs}}{\textit{Productivity}}$
2. Productivity $= \frac{\textit{Total number of adjustable FPs}}{\textit{Effort}}$
3. Cost of a Software $= \textit{Effort} \times \textit{Pay}$
4. Quality of a Software $=$ Number of Errors in function points
5. Development cost per function point $= \frac{\textit{Cost of a Software}}{\textit{Total number of adj. FPs}}$
6. Documentation $= \frac{\textit{Pages per documentation } (P_{pdoc})}{\textit{Function point}}$

Problem-based Estimation (cont.)

Case Study: A software company has delivered a scientific application with following estimate values related to information domain characteristics which include inputs - 15, outputs - 12, inquiries - 10, files - 6, external interfaces - 4. Compute adjustable and unadjustable function points with $\sum f_i = 35$ and the weight factors are treated as 3 as well as complex weight factors. Compute the cost of the software by considering historical data where the productivity of every employee is 4 function points/month and a salary of \$6000. Prepare the report which specifies effort and cost required for the development.

Problem-based Estimation (cont.)

Solution

Information Domain Value	Estimated Value	Weight	FP Count
Number of Inputs	15	3	45
Number of Outputs	12	3	36
Number of Inquiries	10	3	30
Number of Files	6	3	18
Number of Interface	4	3	12
Count total			141

- Number of unadjustable function points = count total =141
- Effort Adjustment Factor (EAF) $= 0.65 + 0.01 \times 35 = 1.0$
- Number of adjustable function points = count total $\times$ EAF $=$ $141 \times 1.0 = 141$
- Productivity $\rightarrow$ 4 function points/month
- Effort $= \frac{\textit{Number of adjustable function points}}{\textit{Productivity}} = \frac{141}{4} = 35$ person-month
- Cost of software development = Effort $\times$ Pay $= 35 \times \$6000$ $= \$210K$

Process-based Estimation

- **Idea**: base the estimate on the process that will be used for software development.
- The process is decomposed into a relatively small set of tasks and the effort required to accomplish each task is estimated.
- Begins with a description of software functions obtained from the project scope.
- A series of framework activities must be performed for each function.

Process-based Estimation (cont.)

	Non-technical Activities (NTA)			Technical Activities				NTA		
Activity	CC	Planning	Risk Analysis	Engineering		Construction Release		CE	Total	Cost
Task				Analysis	Design	Code	Test			
GUI				1.0	2.0	1.0	2.0	n/a	6.0	$30K
DB				2.0	3.0	1.5	3.0	n/a	9.5	$47.5
Server				1.5	2.5	1.5	3.5	n/a	9.0	$45K
Client				1.5	2.0	1.0	3.0	n/a	7.5	$37.5
Totals	0.25	0.25	0.25	6.0	9.5	5.0	11.5		32.75	$160
Effort	0.75%	0.75%	0.75%	18%	29%	15%	35%		100%	
Cost	$1.25K	$1.25K	$1.25K	$30K	$47.5K	$25K	$57.5K		$163.75K	

Table: Process-based estimation table (CC →Customer Communication, and CE → Customer Evaluation)

- Labor rate/Salary: $5000
- Process-based estimation approach provides the microscopic view of effort and cost distribution among modules and activities.

Empirical Estimation Models

- Use of empirically derived formulas to predict effort as a function of LOC or FP.
- The empirical data that support most estimation models are derived from a limited sample of projects.
- No estimation model is appropriate for all classes of softwares and in all development environments.
- An estimation model should be calibrated to reflect local conditions.
- The model should be tested by applying data collected from completed projects, plugging the data into the model, and then comparing actual to predicted results.
- If agreement is poor, the model must be tuned and retested before it can be used.

The Structure of Estimation Models

- The typical estimation model is derived using regression analysis on data collected from past software projects.
- The overall structure of such models takes the form :

$$E = A + B \times (e_v)^C$$

where,

  - A, B, and C are empirically derived constants.
  - E is effort in person-months, and
  - e_v is the estimation variable (either LOC or FP).

- In addition to the above relationship, the majority of estimation models have some form of project adjustment component that enables E to be adjusted by other project characteristics (e.g., problem complexity, staff experience, development environment).

The Structure of Estimation Models (cont.)

- **LOC-oriented estimation models**
  - Walston-Felix Model $\rightarrow E = 5.2 \times (KLOC)^{0.91}$
  - Bailey-Basili Model $\rightarrow E = 5.5 + 0.73 \times (KLOC)^{1.16}$
  - Boehm Simple Model $\rightarrow E = 3.2 \times (KLOC)^{1.05}$
  - Doty model for KLOC>9 $\rightarrow E = 5.288 \times (KLOC)^{1.047}$
- **FP-oriented estimation models**
  - Albrecht and Gaffiney Model $\rightarrow$ E = -91.4 + 0.355 FP
  - Kemerer Model $\rightarrow$ E = -37 + 0.96 FP
  - Small project regression model $\rightarrow$ E = -12.88 + 0.405 FP
- Each estimation model will yield a different result for the same values of LOC and FP.
  $\therefore$ Estimation models must be calibrated for local needs.

The COCOMO Model

- **COnstructive COst MOdel**
- Proposed by Barry Boehm.
- It is empirically derived cost estimation model used for predicting efforts based on LOC.
- Most widely used software cost estimation model in the industry.
- It is a hierarchy of 3 models which includes
  - Basic COCOMO
  - Intermediate COCOMO
  - Advanced COCOMO

The COCOMO Model (cont.)

- **Basic COCOMO**
  - Computes software development efforts and development duration as a function of project size which is expressed as lines of code (LOC).
  - Basic COCOMO provides rough idea about effort needed for project development.

The COCOMO Model (cont.)

- **Intermediate COCOMO**
  - Exact effort can be derived.
  - Computes software development efforts as a function of project size and a set of cost drivers which includes subjective assessment of personnel attributes, project attributes, product attributes, software and hardware attributes.
  - These cost drivers will have major impact so they should be assessed prior to the effort estimation.
  - The impact of these cost drivers ranges from 0.7 to 1.3 (as per textbooks) and from 0.9 to < 2 (as per organizations).

The COCOMO Model (cont.)

- **Advanced COCOMO**
  - Incorporates characteristics of intermediate version along with the impact of cost drivers on software process development as it progresses.

The COCOMO Model (cont.)

Classification of Softwares (by Barry Boehm)

1. **Organic mode software**
   - Application software with no database.
     - Simple requirements,
     - Skills - Low Level,
     - Development duration - in months,
     - Team size - (1 to 50)
   - **Examples** - scientific software, business software, compiler/interpreter, simple inventory analysis, operating system - low end systems (laptop/desktop)

The COCOMO Model (cont.)

2. **Semidetached mode software**
   - Utility software with moderate database.
     - Composite requirements,
     - Skills - mixed (use of more than one language),
     - Development duration - in years,
     - Team size - (50 to 100)
   - **Examples** - Operating system - medium-level transactions, transaction processing system (any system with minimal database), simple command and control systems - space and military applications.

The COCOMO Model (cont.)

3. **Embedded mode software**
   - System software with very large database.
     - Complex requirements,
     - Skills - very high-level skills,
     - Development duration - in years,
     - Tteam size - (100 to 1000)
   - **Examples** - Operating system - high end systems (supercomputer/mainframe), transaction processing system, complex command and control systems (space and military applications).

The COCOMO Model (cont.)

(1) Basic COCOMO

- Purely based on LOC.
- Development Effort: E or $D_E = a_b(KLOC)^{b_b}$ person-month
- Development Duration: D or $D_D = c_b(E)^{d_b}$ months
- Number of programmers/persons: N = $\frac{E}{D}$ persons

The COCOMO Model (cont.)

(2) Intermediate COCOMO

- Based on LOC and Cost Drivers
- Cost Drivers
  - Personnel attributes
    - Average programmer ($<$ 1 year experience, 1 unit of work)
    - Excellent programmer ($>$ 1 year experience, 2 unit of work)
    - Super programmer (computer, 3 unit of work)
  - Project attributes
  - Product attributes
  - Software and hardware attributes

The COCOMO Model (cont.)

- Development Effort: E or D_E = $a_i(KLOC)^{b_i} \times$ *(Effort Adjustment Factor (EAF))* person-month
  $= a_i(KLOC)^{b_i} \times$ *(Cost driver)* person-month
- Development Duration: D or $D_D = c_b(E)^{d_b}$ months
- Number of programmers/persons: N = $\frac{E}{D}$ persons

	Basic COCOMO				Intermediate COCOMO			
	a_b	b_b	c_b	d_b	a_i	b_i	c_b	d_b
Organic Mode	2.4	1.05	2.5	0.38	3.2	1.05	2.5	0.38
Semidetached Mode	3.0	1.12	2.5	0.35	3.0	1.12	2.5	0.35
Embedded Mode	3.6	1.20	2.5	0.32	2.8	1.20	2.5	0.32

The COCOMO Model (cont.)

Case Study: Evaluate the effort, duration and the number of people required for developing all categories of projects of size 24K related to basic COCOMO as well as intermediate COCOMO with the impact of cost drivers in following two cases:

- 1.127
- 1.567

Calculate the cost required for each project by considering a salary of $6000? Prepare the report for all scenarios.

The COCOMO Model (cont.)

Solution

- Project Size (KLOC) =24 K
- Basic COCOMO

Organic mode software

- Development Effort:
  $E = a_b(KLOC)^{b_b} = 2.4(24)^{1.05}$
  = 67.52 person-month
- Development Duration:
  $D = c_b(E)^{d_b} = 2.5(67.52)^{0.38}$
  = 12.39 month
- Number of programmers:
  N= $\frac{E}{D} = \frac{67.52}{12.39}$ =5.44
  ≡ 5 persons

Semidetached mode software

- Development Effort:
  $E = a_b(KLOC)^{b_b} = 3.0(24)^{1.12}$
  = 105.42 person-month
- Development Duration:
  $D = c_b(E)^{d_b} = 2.5(105.42)^{0.35}$
  = 12.76 month
- Number of programmers:
  N= $\frac{E}{D} = \frac{105.42}{12.76}$ =8.26
  ≡ 8 persons

The COCOMO Model (cont.)

Embedded Mode

- Development Effort:
  $E = a_b(KLOC)^{b_b} = 3.6(24)^{1.20}$
  $= 163.13$ person-month
- Development Duration:
  $D = c_b(E)^{d_b} = 2.5(163.13)^{0.32}$
  $= 12.76$ month
- Number of programmers: N$= \frac{E}{D} = \frac{163.13}{12.76} = 12.78$
  $\equiv 13$ persons

The COCOMO Model (cont.)

- **Intermediate COCOMO** (Cost driver 1.127)

Organic mode software

- Development Effort:
  $E = a_i(KLOC)^{b_i} \times$ cost driver
  $= 3.2(24)^{1.05} \times 1.127$
  $=$ 101.46 person-month
- Development Duration:
  $D = c_b(E)^{d_b} = 2.5(101.46)^{0.38}$
  $=$ 14.46 month
- Number of programmers:
  N$= \frac{E}{D} = \frac{101.46}{14.46} = 7.0165$
  $\equiv$ 7 persons

Semidetached mode software

- Development Effort:
  $E = a_i(KLOC)^{b_i} \times$ cost driver
  $= 3.0(24)^{1.12} \times 1.127$
  $=$ 118.81 person-month
- Development Duration:
  $D = c_b(E)^{d_b} = 2.5(118.81)^{0.35}$
  $=$ 13.30 month
- Number of programmers:
  N$= \frac{E}{D} = \frac{118.81}{13.30} = 8.933$
  $\equiv$ 9 persons

The COCOMO Model (cont.)

Embedded Mode

- Development Effort:
  $E = a_i(KLOC)^{b_i} \times$ cost driver $= 2.8(24)^{1.20} \times 1.127$
  $= 142.99$ person-month
- Development Duration:
  $D = c_b(E)^{d_b} = 2.5(142.99)^{0.32}$
  $= 12.23$ month
- Number of programmers: N$= \frac{E}{D} = \frac{142.99}{12.23} = 11.69$
  $\equiv 12$ persons

The COCOMO Model (cont.)

	Basic COCOMO				Intermediate COCOMO (Cost Driver: 1.127)			
	E	D	N	Cost	E	D	N	Cost
Organic Mode	67.52	12.39	5	405K	101.46	14.46	7	608K
Semidetached Mode	105.42	12.76	8	632K	118.81	13.30	9	713K
Embedded Mode	163.13	12.76	13	978K	143	12.23	12	858K

- Here, E: Development Effort, D: Development Duration, N: Team Size.
- Compute the development effort, duration, team size, and cost of development for all modes using intermediate COCOMO with cost driver 1.567.

The Software Equation

- The *software equation* is a dynamic multi-variable empirically derived cost estimation model for predicting effort throughout the life of a software development project.
- The model has been derived from productivity data collected for over 4000 contemporary software projects.
- Based on the data, the estimation model takes the form:

$$E = \frac{LOC \times B^{0.333}}{P^3} \times \frac{1}{t^4} \tag{1}$$

where

- E $\rightarrow$ Effort in person-months or person-years
- t $\rightarrow$ Project duration in months or years
- B $\rightarrow$ Special skill factor
- P $\rightarrow$ Productivity parameter

The Software Equation (cont.)

- The productivity parameter reflects
  - overall process maturity and management practices,
  - the extent to which good software engineering practices are used,
  - the level of programming languages used,
  - the state of the software environment,
  - the skills and experience of the software team, and
  - the complexity of the application.

Software	Productivity (P)
Real-time Embedded Software	2000
Network-based Software	10000
Scientific Software	12000
Business System Applications	28000

Table: Typical values of productivity (P) for different categories of software.

- The productivity parameter can be derived for local conditions using historical data collected from past development projects.

The Software Equation (cont.)

- Software equation has two independent parameters:
  1. an estimate of size (in LOC) and
  2. an indication of project duration in calender months or years.
- The special skill factor (B) increases slowly as "need for integration, testing, quality assurance, documentation, and management skills grows."
  - For small programs (KLOC = 5 to 15), B = 0.16.
  - For programs greater than 70 KLOC, B = 0.39.

Putnam and Myers Model

- Based on software equation, Putnam and Myers proposed set of equations for evaluating *minimum development time* for a software as well as *development effort*.
- Minimum development time is defined as

$$t_{min} = 8.14 \times \frac{LOC}{P^{0.43}} \text{ in months for } t_{min} \geq 6 \text{ months}$$

- Development effort is defined as

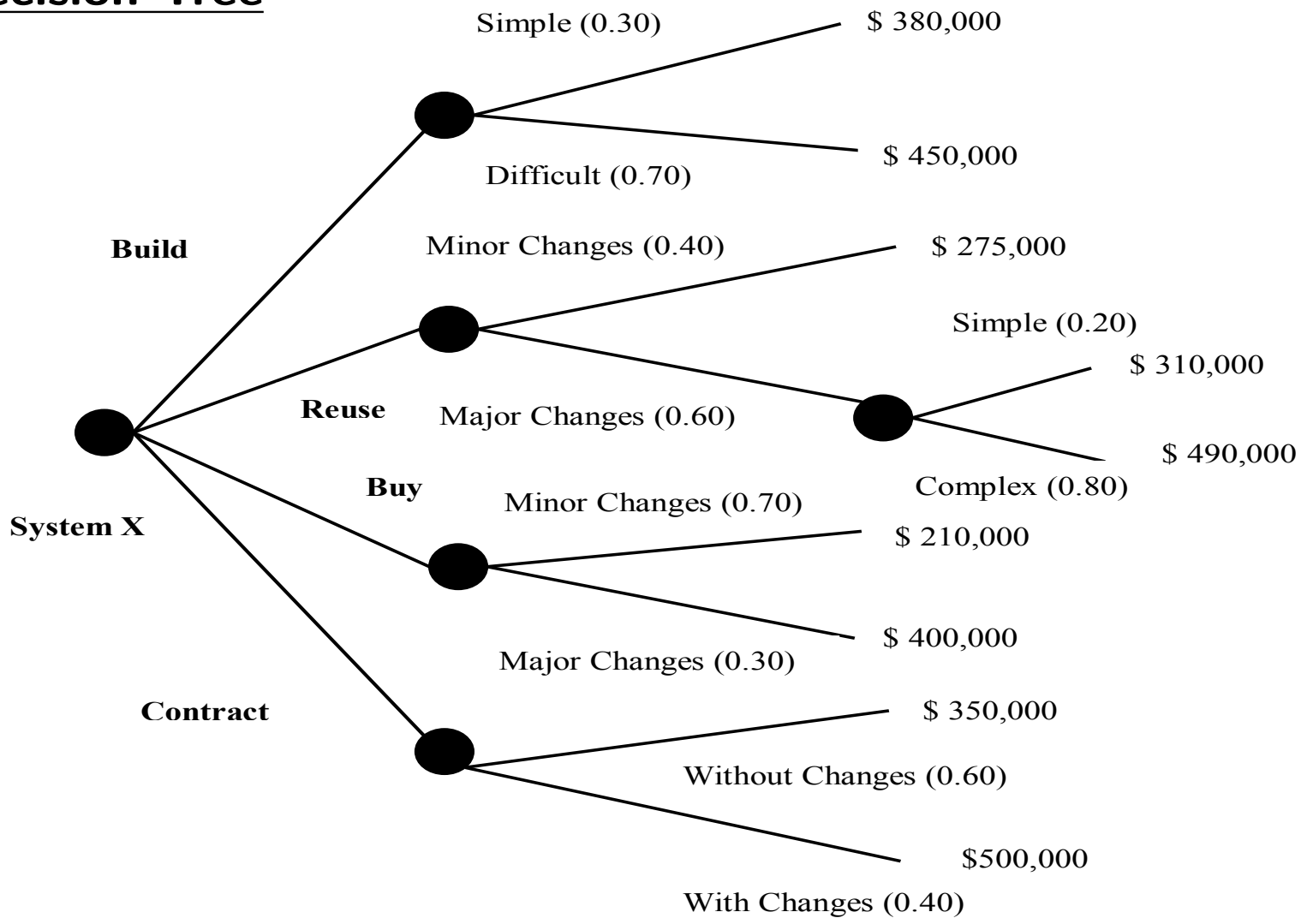
$$E = 180 \times Bt^3 \text{ in person-months for } E \geq 20 \text{ person-months}$$

The Make/Buy Decision

- Cost effectiveness: acquiring a software vs developing a software.
- The make/buy decision is made based on following conditions:
  - Will the delivery date of software product be sooner than that for 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?

The Make/Buy Decision (cont.)

Decision Tree



The Make/Buy Decision (cont.)

- Decision Tree: most widely adapted tool used during planning stage to perform critical decisions based on resources required for product development.
- Used only if more than one option is concerned to the given resource.
- Using decision tree, an estimated value (expected cost) is determined for every option based on two parameters:
  1. Path probability
  2. Estimated cost

  $Ev_{Option} = \sum(\text{Path Probability} \times \text{Estimated Cost})$
- After evaluating all the options, whichever is cost effective in nature will be selected for project development.

The Make/Buy Decision (cont.)

- Expected $Cost_{build} = 0.30(\$380K) + 0.70(\$450K) = \$429K$
- Expected $Cost_{reuse} =$
  $0.40(\$275K) + 0.60[0.20(\$310K) + 0.80(\$490K)] = \$382K$
- Expected $Cost_{buy} = 0.70(\$210K) + 0.30(\$400K) = \underline{\$267K}$
- Expected $Cost_{contract} =$
  $0.60(\$350K) + 0.40(\$500K) = \$410K$