SWE 402: Software Engineering Economics

By

E. K. Olatunji

Software Engineering Programmme
FCAS
Thomas Adewumi University, Oko

April 2024

Course Description

- Software engineering economics is the application of economic principles and techniques to the analysis, design, implementation, and maintenance of software systems.
- It involves a systematic approach to software development that takes into account the economic factors involved in software projects, such as cost, schedule, risk, and quality.
- Software engineering economics aims to provide a rational basis for decision-making in software engineering, by evaluating the costs and benefits of various software development alternatives.
- This involves assessing the trade-offs between the various software development strategies, such as using open-source software versus proprietary software, or outsourcing software development versus in-house development

Course Contents

- Software engineering economics fundamentals; lifecycle economics; Risk and uncertainty- goals, estimates and plans, estimation techniques, addressing uncertainty, prioritization, decisions under risk and uncertainty; Economic analysis methods – for- profit decision analysis, minimum acceptable rate of return, return on investment and capital employed, cost-benefit analysis, cost-effectiveness analysis, break-even analysis, business case, multiple attribute evaluation, and optimization analysis; Practical considerations – the “good enough” principle, friction – free economy, ecosystems, and offshoring and outsourcing.

Course Objectives

- To explain important terms relevant to the course title
- To explain the relevance of the course to software engineers
- To describe important economic techniques and principles that are applicable to SWE process
- To discuss software sizing and estimation
- To describe function point Analysis (FPA) & its practical application to software engineering process
- To describe the COCOMO model and its practical application to swe process

Learning Objectives

- **At the end of the course students should be able to**:
- 1. explain 4 technical terms related to the course title
- 2. list 5 importance of SWEE to a Software Engineers
- 3. List and explain 5 important economic techniques applicable to software process
- 4. calculate the adjusted function points for a given problem
- 5. Explain the concept of COCOMO model of software costing
- 6. Calculate the effort, time, productivity required for development of a software product
- etc

Course content Sequencing

S/N	SWE 402	Software Engineering Economics (SWEE)	No of hours
1	Week 1-2	Introduction to imp terms in SWEE	
2	Week 3-4	Important Economic terms relevant to SWE	
3	Week 5-6	Software Sizing and Estimation Techniques	
4	Week 7	CA1	
5	Week 8-9	Function Point Analysis & its application	
6	Week 9-10	COCOMO model	
7	Week 11	CA 2	
8	Week 12		

Reference Materials

- 1. Software Engineering – A Practitioners Approach by Roger S. Pressman, 5th edition upwards
- 2. Software Engineering by Ian Sommervillie, 8th edition upwards
- 3. SWE Economics Lecture series, by Riaz Ahmad, Dept of Computer Sc., COMSATs University, Islamabad, Pakistan; Available online, Retrieved April 2024
- 4. Software Engineering Economics by Barry Boehm
- 5. Online Resources

Reading List

- READING LIST:
- Software Engineering Economics by Barry Boehm
- Managing Software Requirements: A Unified Approach by Dean Leffingwell and Don
- The Economics of Software Quality by Capers Jones and Olivier Bonsignour -.
- Software Estimation: Demystifying the Black Art by Steve McConnell –
- Software Cost Estimation with Cocomo II by Barry Boehm -

SWE 402 Lecture Note Draft1b

SWE 402 Software Engineering Economics – Introduction2

Introduction to SWE Economics – Concepts and Terminologies -

= =

What is software Engineering?

Software engineering is the systematic application of engineering principles, methodologies, and practices to the development, maintenance, and evolution of software systems. It encompasses a broad range of activities, including requirements analysis, design, implementation, testing, deployment, and maintenance, with the goal of producing high-quality software products that meet user needs, are reliable, scalable, and maintainable.

One widely accepted definition of software engineering is provided by the IEEE Computer Society, which defines software engineering as "the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software." (IEEE Computer Society, 1993)

Reference:

- IEEE Computer Society. (1993). IEEE Standard Glossary of Software Engineering Terminology. IEEE.

= = =

Examples of Engineering Principles

Engineering principles serve as fundamental guidelines that govern the design, development, and implementation of various systems, structures, and processes. Here are some examples of engineering principles:

1. **Abstraction**: The process of simplifying complex systems by focusing on the most relevant aspects while hiding unnecessary details. Abstraction allows engineers to manage complexity and develop scalable solutions.

2. **Modularity**: Breaking down complex systems into smaller, manageable modules or components that can be developed, tested, and maintained independently. Modularity promotes reusability, flexibility, and ease of maintenance.

3. **Optimization**: Finding the best possible solution given constraints and objectives. Optimization involves balancing trade-offs and making decisions to maximize efficiency, performance, or other desired criteria.

4. **Safety**: Prioritizing the safety and well-being of users, workers, and the environment in the design and operation of engineering systems. Safety principles include risk assessment, hazard mitigation, and compliance with safety regulations and standards.

5. **Reliability**: Ensuring that engineering systems consistently perform their intended functions under specified conditions for a specified period. Reliability principles involve designing for redundancy, fault tolerance, and robustness to minimize the likelihood of failures.

6. **Sustainability**: Designing and implementing engineering solutions that meet present needs without compromising the ability of future generations to meet their own needs. Sustainability principles consider environmental, social, and economic impacts over the lifecycle of a system.

7. **Interdisciplinary Collaboration**: Engaging with professionals from diverse fields to leverage their expertise and perspectives in solving complex engineering problems. Interdisciplinary collaboration promotes innovation and holistic problem-solving approaches.

8. **Ethics**: Upholding ethical standards and values in engineering practice, including honesty, integrity, fairness, and respect for the well-being of stakeholders and society at large. Ethical principles guide decision-making and behavior in engineering professions.

9. **Lifecycle Management**: Considering the entire lifecycle of engineering systems, from conception and design to manufacturing, operation, maintenance, and disposal. Lifecycle management principles aim to optimize performance, cost, and environmental impact over the system's lifespan.

10. **Continuous Improvement**: Embracing a culture of continuous learning, innovation, and adaptation to drive ongoing improvements in engineering processes, products, and services. Continuous improvement principles involve feedback mechanisms, performance measurement, and process optimization.

These engineering principles serve as foundational concepts that guide engineers in their work and contribute to the development of safe, reliable, sustainable, and innovative solutions to complex challenges.

= =

What is Economics?

Economics is the social science that studies how individuals, businesses, governments, and societies allocate resources to satisfy their needs and wants. It examines how decisions are made in the face of scarcity, where unlimited wants and needs meet limited resources. Economics analyzes various factors such as production, distribution, consumption, and the behavior of markets and individuals to understand and explain economic phenomena.

One authoritative reference for the definition of economics is provided by Nobel laureate economist Paul Samuelson, who defines economics as "the study of how societies use scarce resources to produce valuable commodities and distribute them among different people." (Samuelson & Nordhaus, 2010)

Reference:

- Samuelson, P. A., & Nordhaus, W. D. (2010). Economics. McGraw-Hill Education.

= = =

What is Software Engineering Economics (SWEE)?

Software engineering economics is the application of economic principles and techniques to the analysis, design, development, deployment, and maintenance of software systems. It encompasses various aspects related to the financial management of software projects and the optimization of resources to achieve desired outcomes within budgetary constraints.

In essence, software engineering economics addresses questions such as:

1. **Cost Estimation**: How much will it cost to develop and maintain a software system? This involves estimating both tangible costs (e.g., hardware, software licenses) and intangible costs (e.g., developer time, opportunity costs).

2. **Risk Management**: What are the risks associated with a software project, and how can they be managed to minimize their impact on cost and schedule?

3. **Return on Investment (ROI)**: What is the expected return on investment from developing a software system? This involves comparing the financial benefits of a project (e.g., increased revenue, cost savings) to the costs incurred during its lifecycle.

4. **Resource Allocation**: How should resources such as time, money, and personnel be allocated to maximize the value delivered by software projects?

5. **Lifecycle Costing**: How do costs vary throughout the lifecycle of a software system, from initial development to ongoing maintenance and support? Understanding lifecycle costs helps in making informed decisions about resource allocation and budget planning.

6. **Quality Management**: How can the quality of software be measured and managed to minimize rework and ensure customer satisfaction, while balancing cost constraints?

7. **Value-Based Decision Making**: How can decisions about software projects be made based on the value they deliver to stakeholders? This involves prioritizing features and enhancements that provide the highest value to customers and align with business objectives.

Overall, software engineering economics provides a framework for making rational, data-driven decisions throughout the software development lifecycle, with the ultimate goal of delivering high-quality software products within budget and on schedule.

= = =

= =

Examples of economic principles that can be applied in software engineering with reference

Certainly, here are examples of economic principles applied in software engineering, along with relevant references:

1. **Supply and Demand**: The principle of supply and demand helps in understanding the market dynamics for software products and services. For instance, the demand for software developers may exceed the supply, leading to higher wages and competition for talent. This principle is discussed in "Economics" by McConnell, Brue, and Flynn (2020).

2. **Opportunity Cost**: Opportunity cost refers to the value of the next best alternative forgone when a decision is made. In software engineering, this principle can be applied when choosing between different development technologies or project opportunities. Deciding to use one programming language over another involves considering the opportunity cost of not using the alternative. This concept is explored in "Microeconomics" by Perloff (2018).

3. **Marginal Analysis**: Marginal analysis involves examining the additional costs and benefits associated with small changes in production or consumption. In software engineering, marginal analysis can be used to determine the optimal level of resources to allocate to a project. For example, it helps in deciding whether to invest additional time in optimizing code for marginal performance gains. This principle is discussed in "Principles of Economics" by Mankiw (2018).

4. **Economies of Scale**: Economies of scale occur when the average cost of production decreases as the quantity produced increases. In software engineering, this principle applies when developing software products for mass distribution. The cost per unit of software decreases as the number of units produced increases. This concept is explained in "Managerial Economics & Business Strategy" by Baye and Prince (2017).

5. **Time Value of Money**: The time value of money principle recognizes that a dollar today is worth more than a dollar in the future due to its potential earning capacity. In software engineering, this principle is applied when estimating the present value of future cash flows associated with a project. Discounted cash flow analysis is used to evaluate the financial feasibility of software investments. This principle is discussed in "Foundations of Financial Management" by Block, Hirt, and Danielsen (2019).

By applying these economic principles, software engineers and managers can make more informed decisions regarding resource allocation, project prioritization, and investment strategies in the software development process. These principles are widely recognized in economic literature and provide a solid foundation for understanding the economic aspects of software engineering.

= = =

What is the relevance of software engineering economics?

The relevance of software engineering economics lies in its ability to provide a systematic approach to managing software development projects and optimizing resources. Here are some key points highlighting its significance:

1. **Cost Control**: Software engineering economics helps in estimating, managing, and controlling the costs associated with software development. By understanding the cost drivers and employing cost estimation techniques, organizations can allocate resources effectively and prevent budget overruns.

2. **Risk Management**: Software projects are inherently risky due to uncertainties in requirements, technology, and other factors. Software engineering economics provides methodologies for identifying, assessing, and mitigating risks, thus increasing the likelihood of project success.

3. **ROI Maximization**: By applying economic principles, organizations can prioritize projects based on their potential return on investment (ROI). Software engineering economics assists in evaluating the financial benefits of software projects against their costs, enabling decision-makers to invest resources in projects with the highest ROI.

4. **Resource Allocation**: Efficient resource allocation is crucial for delivering software projects on time and within budget. Software engineering economics helps in allocating resources such as time, money, and personnel optimally, thereby maximizing productivity and minimizing waste.

5. **Quality Assurance**: Quality is a key determinant of the success of software projects. Software engineering economics emphasizes the importance of balancing quality requirements with cost constraints. By investing in quality assurance measures upfront, organizations can reduce rework costs and enhance customer satisfaction.

6. **Lifecycle Costing**: Understanding the total cost of ownership (TCO) of software systems over their lifecycle is essential for making informed decisions. Software engineering economics enables organizations to assess the long-term costs of software development, maintenance, and support, facilitating better budget planning and resource management.

7. **Strategic Decision Making**: Software engineering economics provides a framework for strategic decision-making regarding software investments and project prioritization. By considering factors such as market demand, competitive landscape, and technological advancements, organizations can align their software development efforts with business goals and objectives.

In summary, software engineering economics is relevant because it helps organizations make informed decisions, manage risks, control costs, and maximize the return on investment from software projects. By applying economic principles to software development, organizations can enhance their competitiveness, innovation, and overall business performance.

= =

4**. Why does one need to study software Engineering economics with reference?**

Studying software engineering economics with reference to economic principles and theories is essential for several reasons:

1. **Resource Optimization**: Understanding software engineering economics enables organizations to optimize resources such as time, money, and personnel. By applying economic principles, decision-makers can allocate resources efficiently to maximize productivity and minimize waste.
2. **Cost Control**: Software projects often involve significant financial investments. Studying software engineering economics helps in estimating, managing, and controlling costs throughout the software development lifecycle, preventing budget overruns and ensuring financial viability.
3. **Risk Management**: Software projects are inherently risky due to uncertainties in requirements, technology, and market dynamics. Software engineering economics provides methodologies for identifying, assessing, and mitigating risks, thereby increasing the likelihood of project success.
4. **ROI Maximization**: By analyzing the costs and benefits of software projects, organizations can prioritize investments based on their potential return on investment (ROI). Studying software engineering economics helps in evaluating the financial viability of projects and selecting those with the highest ROI.
5. **Strategic Decision Making**: Software engineering economics provides a framework for strategic decision-making regarding software investments, project prioritization, and resource allocation. Understanding economic principles helps decision-makers align software development efforts with business goals and objectives.
6. **Quality Assurance**: Balancing quality requirements with cost constraints is crucial for the success of software projects. Studying software engineering economics helps organizations invest in quality assurance measures upfront, reducing rework costs and enhancing customer satisfaction.

= = =

1. **Examples of economic techniques that can be applied in software engineering with reference**

Certainly! Here are examples of economic techniques commonly applied in software engineering, along with relevant references:

1. **Cost-Benefit Analysis (CBA)**: CBA is a technique used to compare the costs and benefits of a project or decision. In software engineering, CBA helps in evaluating the financial feasibility of software projects by quantifying the expected benefits against the costs involved. This technique is discussed in "Cost-Benefit Analysis: Concepts and Practice" by Boardman, Greenberg, Vining, and Weimer (2017).

2. **Return on Investment (ROI) Analysis**: ROI analysis calculates the return on investment generated by a project relative to its costs. In software engineering, ROI analysis is used to assess the profitability of software investments and prioritize projects with the highest ROI potential. This technique is discussed in "Return on Investment in Training and Performance Improvement Programs" by Phillips and Phillips (2016).

3. **Discounted Cash Flow (DCF) Analysis**: DCF analysis calculates the present value of future cash flows associated with a project by discounting them back to their current value. In software engineering, DCF analysis helps in evaluating the financial viability of software projects by considering the time value of money. This technique is discussed in "Valuation: Measuring and Managing the Value of Companies" by McKinsey & Company Inc. (2015).

4. **Cost Estimation Techniques**: Various techniques are used to estimate the costs of software development projects, such as expert judgment, analogy-based estimation, and algorithmic models like COCOMO (Constructive Cost Model) and function point analysis. These techniques help in predicting project costs based on factors such as project size, complexity, and team productivity. They are discussed in "Software Estimation: Demystifying the Black Art" by McConnell (2006).

5. **Risk Analysis and Management**: Risk analysis techniques, such as probability analysis, impact analysis, and risk prioritization, are used to identify, assess, and mitigate risks associated with software projects. Risk management helps in minimizing the potential negative impacts of risks on project cost, schedule, and quality. These techniques are discussed in "Managing Software Requirements: A Use Case Approach" by Wiegers and Beatty (2013).

6. **Earned Value Management (EVM)**: EVM is a technique used to track the progress and performance of a project by integrating measurements of scope, schedule, and cost. In software engineering, EVM helps in monitoring project progress, identifying variances from the planned budget, and forecasting future project costs. This technique is discussed in "Performance-Based Earned Value" by Humphreys (2007).

By applying these economic techniques, software engineers and project managers can make more informed decisions regarding resource allocation, project planning, and risk management, ultimately leading to more successful software projects. These techniques are widely used in both academia and industry and have been proven effective in optimizing the economic aspects of software engineering

= =

Software Engineering Economics Software Sizing And Estimation

(Slightly Modified by E. K. Olatunji, TAU, Oko on 22/04-2024)

Instructor Name: Riaz Ahmad

Department of Computer Science, COMSATS University Islamabad, Wah Campus, Pakistan

Outline

- What is software Sizing and Estimation
- Importance of Software Sizing
- Software Sizing Techniques
- Advantages and disadvantages of approaches
- Sizing Steps
- Sizing Standards
- Summary of the lecture
- What next……..

What is Software Sizing or Software size estimation

Software sizing or Software size estimation is an activity in software engineering that is used to determine or estimate the size of a software application or component in order to be able to implement other software project management activities”

Importance of Software Sizing and Estimation

Software project management is the process of planning, implementing, and monitoring, controlling, leading and managing software projects.

It is a subdiscipline of project management

For the successful completion of software development process, perfect planning is necessary.

Size is the base factor to determine effort, duration, schedule, cost.

Project size Estimation Techniques/Approaches

- Estimation of the size of software is an essential part of Software Project Management.
- It helps the project manager to further predict the effort and time which will be needed to build the project.
- Various measures are used in project size estimation. Some of these are:

1. Lines of Code (or Source lines of code – SLOC)
2. Number of entities in ER diagram
3. Total number of processes in detailed data flow diagram
4. Function points

Project size Estimation Techniques/Approaches

- **Lines of Code (LOC):** As the name suggest, LOC count the total number of lines of source code in a project. The units of LOC are:
- KLOC- Thousand lines of code
- NLOC- Non comment lines of code
- KDSI- Thousands of delivered source instruction
- The size is estimated by comparing it with the existing systems of same kind. The experts use it to predict the required size of various components of software and then add them to get the total size.
- SLOC is typically used to predict the amount of effort that will be required to develop a program, as well as to estimate programming productivity or maintainability once the software is produced.

Project size Estimation Techniques/Approaches

- **Advantages:**

1. Universally accepted and is used in many models like COCOMO.
2. Estimation is closer to developer's perspective.
3. Simple to use.

- **Disadvantages:**

1. Different programming languages contains different number of lines.
2. No proper industry standard exist for this technique.
3. It is difficult to estimate the size using this technique in early stages of project.

Project size Estimation Techniques/Approaches

2. Number of entities in ER diagram: ER model provides a static view of the project. It describes the entities and its relationships. The number of entities in ER model can be used to measure the estimation of size of project. Number of entities depends on the size of the project. This is because more entities needed more classes/structures thus leading to more coding.

- **Advantages:**

1. Size estimation can be done during initial stages of planning.
2. Number of entities is independent of programming technologies used.

- **Disadvantages:**

1. No fixed standards exist. Some entities contribute more project size than others.
2. Just like FPA, it is less used in cost estimation model. Hence, it must be converted to LOC.

More on ER

- An Entity Relationship (ER) Diagram is a type of flowchart that illustrates how “entities” such as people, objects or concepts relate to each other within a system.
-
- ER Diagrams are most often used to design or debug relational databases in the fields of software engineering, business information systems, education and research.
- Also known as ERDs or ER Models, they use a defined set of symbols such as rectangles, diamonds, ovals and connecting lines to depict the interconnectedness of entities, relationships and their attributes. They mirror grammatical structure, with entities as nouns and relationships as verbs.

- Entity

- A definable thing—such as a person, object, concept or event—that can have data stored about it. Think of entities as nouns. Examples: a customer, student, car or product. Typically shown as a rectangle.

Project size Estimation Techniques/Approaches

3. Total number of processes in detailed data flow diagram: Data Flow Diagram(DFD) represents the functional view of a software. The model depicts the main processes/functions involved in software and flow of data between them. Utilization of number of functions in DFD to predict software size. Already existing processes of similar type are studied and used to estimate the size of the process. Sum of the estimated size of each process gives the final estimated size.

- **Advantages:**

1. It is independent of programming language.
2. Each major processes can be decomposed into smaller processes. This will increase the accuracy of estimation

- **Disadvantages:**

1. Studying similar kind of processes to estimate size takes additional time and effort.
2. All software projects are not required to construction of DFD.

Standards for the measurement of software size

CISQ has published two standards for automating the measurement of software size:

•**Automated Function Points** measure the functional size of software.

•**Automated Enhancement Points** measure the size of both functional and non-functional code in one measure

The Automated Function Points measure was specified to mirror the counting guidelines of the International Function Points User Group (IFPUG) as closely as possible, while removing ambiguity to support automation. Traditionally, Function Points have been measured manually and counts can vary by +/-10% among certified counters.

Limitation of using FP

- Function Points have been difficult to use during maintenance and enhancement activities because they do not measure non-functional code, which can account for over half the size of modern applications.

- Automated Enhancement Points were defined to solve this challenge by measuring the size of both the functional and non-functional code and summing them into a Function Point-like measure.

More on CISQ & Function Point (addendum)

- What is CISQ?
- **The Consortium for IT Software Quality** (CISQ) is an IT industry group comprising IT executives from the Global 2000, systems integrators, outsourced service providers, and software technology vendors committed to making improvements in the quality of IT application software.
- Also called ***Consortium for Info & Software Quality***
- In September 2012, CISQ published its standard measures for evaluating and benchmarking the reliability, security, performance efficiency, and maintainability of IT software
-
- The **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 are used to compute a functional size measurement (FSM) of software**. The cost (in dollars or hours) of a single unit is calculated from past projects.

- **FP (Function Point)**
- Function points measure the size of an application system based on the functional view of the system. The size is determined by counting the number of inputs, outputs, queries, internal files and external files in the system and adjusting that total for the functional complexity of the system

Contrasting function points and Lines of code (LOC)

- The use of function points in favor of lines of code seek to address several additional issues:
- The risk of "inflation" of the created lines of code, and thus reducing the value of the measurement system, if developers are incentivized to be more productive. FP advocates refer to this as measuring the size of the solution instead of the size of the problem.
- Lines of Code (LOC) measures reward low level languages because more lines of code are needed to deliver a similar amount of functionality to a higher level language.[7] C. Jones offers a method of correcting this in his work.[8]
- LOC measures are not useful during early project phases where estimating the number of lines of code that will be delivered is challenging. However, Function Points can be derived from requirements and therefore are useful in methods such as estimation by proxy.

CORE METRICS CATEGORIES

https://galorath.com/10-step-software-size-estimation-process/

SOFTWARE SIZE ESTIMATION: THE 10 STEP SOFTWARE ESTIMATION PROCESS

- Ideally, at a minimum the following attributes of a software project would be measured:

1. Cost, in terms of staff effort, phase effort and total effort
2. Defects found or corrected, and the effort associated with them
3. Process characteristics such as development language, process model and technology
4. Project dynamics including changes or growth in requirements or code and schedule
5. Project progress (measuring performance against schedule, budget, etc.)
6. Software structure in terms of size and complexity

Software Sizing Steps

A software estimation process that is integrated with the software development process can help projects establish realistic and credible plans to implement the project requirements and satisfy commitments.

1. STEP ONE: ESTABLISH ESTIMATE SCOPE AND PURPOSE
2. STEP TWO: ESTABLISH TECHNICAL BASELINE, GROUND RULES, AND ASSUMPTIONS
3. STEP THREE: COLLECT DATA
4. STEP FOUR: SOFTWARE SIZING
5. STEP FIVE: PREPARE BASELINE ESTIMATE
6. STEP SIX: QUANTIFY RISKS AND RISK ANALYSIS
7. STEP SEVEN: ESTIMATE VALIDATION AND REVIEW
8. STEP EIGHT: GENERATE A PROJECT PLAN
9. STEP NINE: DOCUMENT ESTIMATE AND LESSONS LEARNED
10. STEP TEN: TRACK PROJECT THROUGHOUT DEVELOPMENT

Objectives of Functional Point Analysis

1. **Encourage Approximation:** FPA helps in the estimation of the work, time, and materials needed to develop a software project. Organizations can plan and manage projects more accurately when a common measure of functionality is available.

2. **To assist with project management:** Project managers can monitor and manage software development projects with the help of FPA. Managers can evaluate productivity, monitor progress, and make well-informed decisions about resource allocation and project timeframes by measuring the software's functional points.

3. **Comparative analysis:** By enabling benchmarking, it gives businesses the ability to assess how their software projects measure up to industry standards or best practices in terms of size and complexity. This can be useful for determining where improvements might be made and for evaluating how well development procedures are working.

Objectives of Functional Point Analysis

.

4 • **Improve Your Cost-Benefit Analysis:** It offers a foundation for assessing the value provided by the program concerning its size and complexity, which helps with cost-benefit analysis. Making educated judgements about project investments and resource allocations can benefit from having access to this information.

5. • **Comply with Business Objectives:** It assists in coordinating software development activities with an organization’s business objectives. It guarantees that software development efforts are directed toward providing value to end users by concentrating on user-oriented functionality.

Benefits of Functional Point Analysis

2. **Technological Independence:** It calculates a software system's functional size independent of the underlying technology or programming language used to implement it. As a result, it is a technology-neutral metric that makes it easier to compare projects created with various technologies.
3. **Better Accurate Project Estimation:** It helps to improve project estimation accuracy by measuring user interactions and functional needs. Project managers can improve planning and budgeting by using the results of the FPA to estimate the time, effort and resources required for development.
4. **Improved Interaction:** It provides a common language for business analysts, developers, and project managers to communicate with one another and with other stakeholders. By communicating the size and complexity of software in a way that both technical and non-technical audiences can easily understand this helps close the communication gap.

3. • **Making Well-Informed Decisions:** FPA assists in making well-informed decisions at every stage of the software development life cycle. Based on the functional requirements, organizations can use the results of the FPA to make decisions about resource allocation, project prioritization, and technology selection.

4. • **Early Recognition of Changes in Scope**: Early detection of changes in project scope is made easier with the help of FPA. Better scope change management is made possible by the measurement of functional requirements, which makes it possible to evaluate additions or changes for their effect on the project's overall size.

Characteristics of Functional Point Analysis

- We calculate the functional point with the help of the number of functions and types of functions used in applications. These are classified into five types:

Measurement Parameters	Examples
Number of External Inputs (EI)	Input screen and tables
Number of External Output (EO)	Output screens and reports
Number of external inquiries (EQ)	Prompts and interrupts
Number of internal files (ILF)	Databases and directories
Number of external interfaces (EIF)	Shared databases and shared routines

THANK YOU

Software Engineering Economics Function Point Analysis

Slightly Modified by E. K. Olatunji, TAU, Oko on 07-05-2024)

Instructor Name: Riaz Ahmad

Department of Computer Science, COMSATS University Islamabad, Wah Campus, Pakistan

Quotation of Today (Synonymous)

- Power begins from within.’
- ‘Ideas are generated in the unconscious.’
- ‘If you focus deeply on a subject, you understand it better and better and more layers of it are revealed to you.’

Outline

- What is Function point (Introduction)
- Objectives of function Point Measurements
- Benefits of FPA
- Components of FPA
- Overview of FPA
- An Example
- Limitation of FP
- Feature Points
- Summery of the lecture
- What next……..

What is Function Point Analysis (Introduction)

Abbreviated as FPA, functional point analysis is one of the mostly preferred and widely used estimation technique used in the software engineering. ... The functional size of the product is measured in the terms of the function point, which is a standard of measurement to measure the software application.

Function point metrics, developed by Alan Albercht of IBM, were first published in 1979 In 1984, the International Function Point Users Group (IFPUG) was set up to clarify the rules, set standards, and promote their use and evolution.

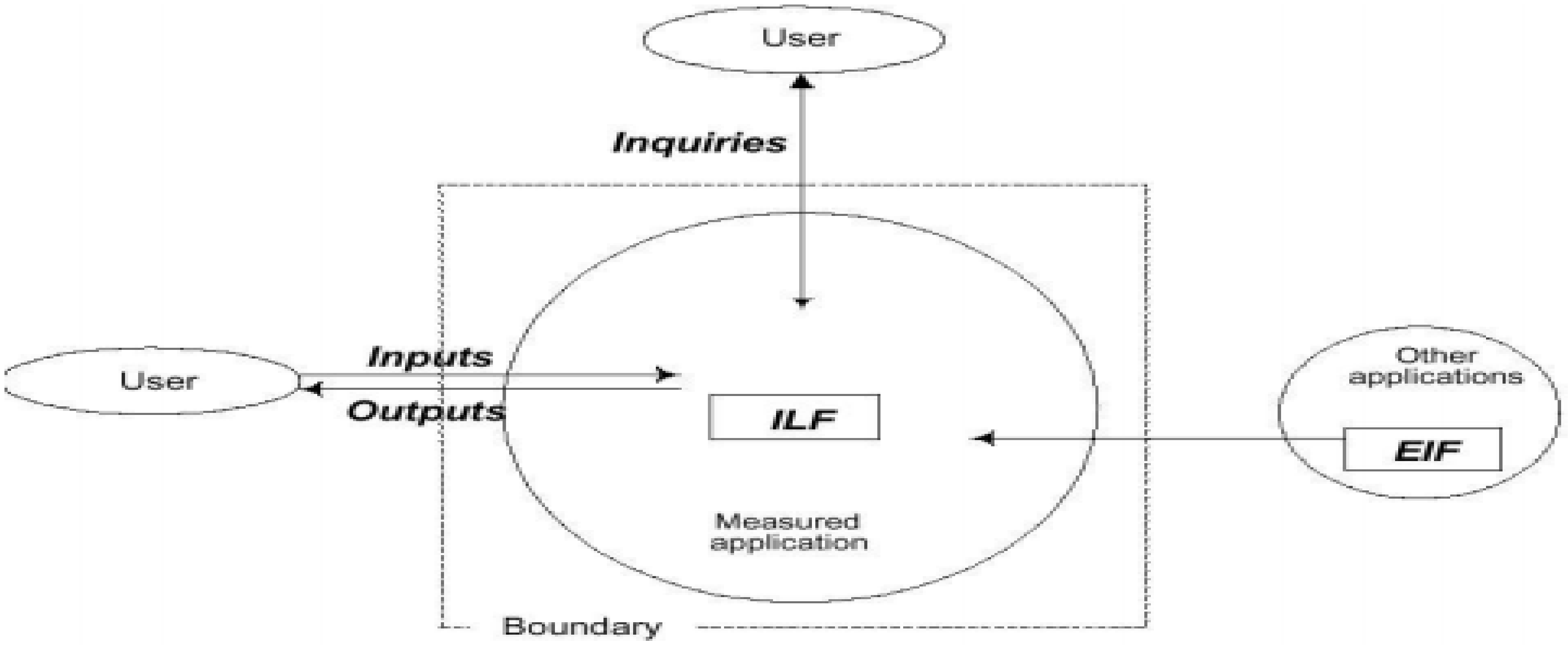
Objectives of Function Point Measurement

1. Measure functionality that the user requests and receives
2. Measure software development and maintenance independently of technology used for implementation

Benefits of FPA

- Organizations can apply function point analysis as:
  - A tool to determine the size of a purchased application package by counting all the functions included in the package
  - A tool to help users determine the benefit of an application package to their organization by counting functions that specifically match their requirements
  - A tool to measure the units of a software product to support quality and productivity analysis
  - A vehicle to estimate cost and resources required for software development and maintenance
  - A normalization factor for software comparison

Components of FPA



Components of FPA

1. **Internal Logical File (ILF):** a user identifiable group of logically related data or control information maintained within the boundary of the application
2. **External Interface File (EIF):** a user identifiable group of logically related data or control information referenced by the
   application, but maintained within the boundary of another application.
3. **External Input (EI):** An EI processes data or control information that comes from outside the application’s boundary
4. **External Output (EO):** An EO is an elementary process that generates data or control information sent outside the
   application’s boundary
5. **External Inquiry (EQ):** An EQ is an elementary process made up of an input-output combination that results in data retrieval

Type of FP Attributes

Measurements Parameters	Examples
1.Number of External Inputs(EI)	Input screen and tables
2. Number of External Output (EO)	Output screens and reports
3. Number of external inquiries (EQ)	Prompts and interrupts.
4. Number of internal files (ILF)	Databases and directories
5. Number of external interfaces (EIF)	Shared databases and shared routines.

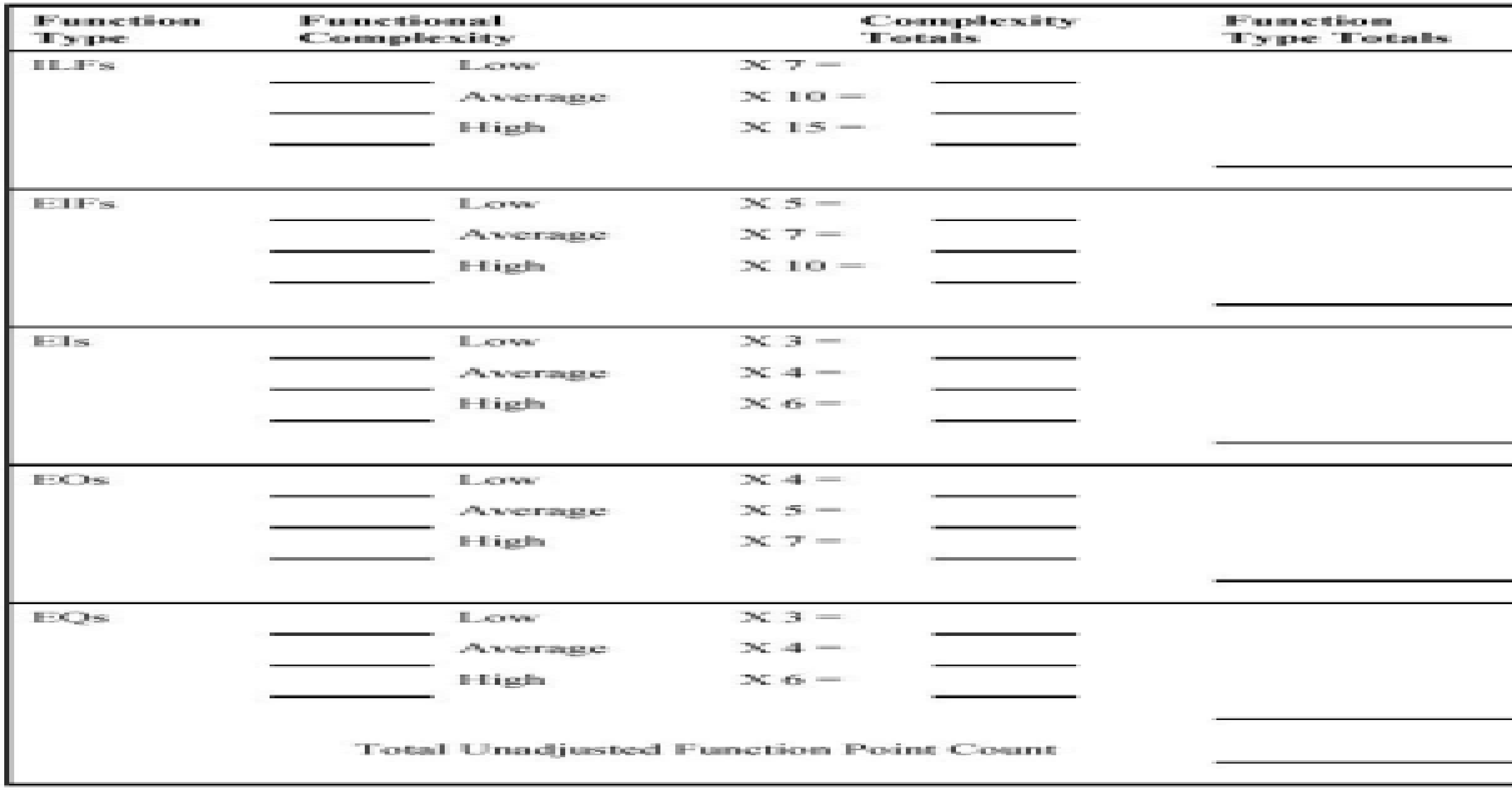
Overview of the FPA

These 5 function types are then ranked

according to their complexity: Low, Average or High, using a set of prescriptive standards. Organizations that use FP methods, develop criteria for determining whether a particular entry is Low, Average or High. Nonetheless, the determination of complexity is somewhat subjective.

After classifying each of the five function types, the UFP is computed using predefined weights for each function type

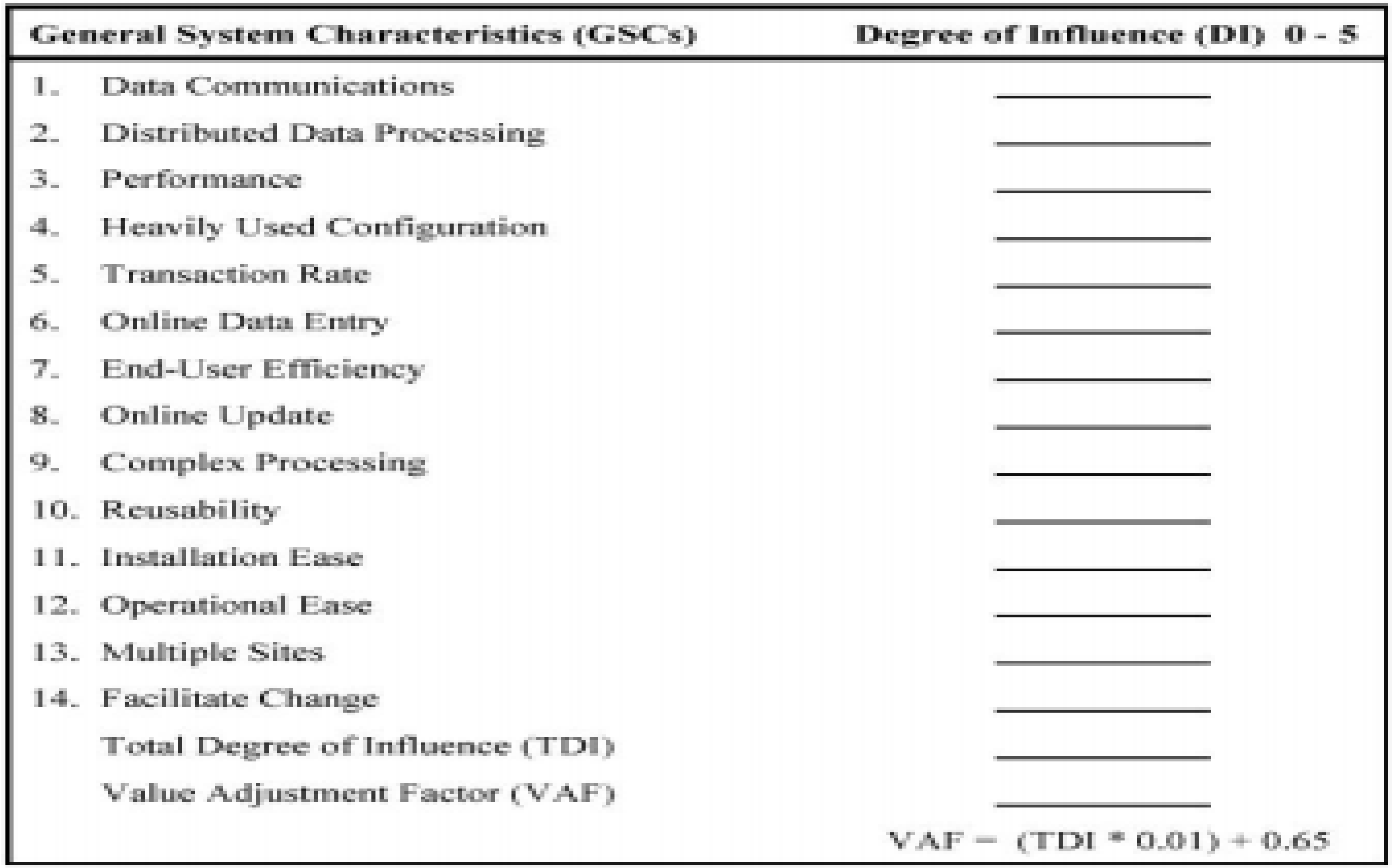
UFP Calculation Table



Function Type	Functional Complexity			Complexity Totals	Function Type Totals
ILFs		Low	X 7 =		
		Average	X 10 =		
		High	X 15 =		
EIFs		Low	X 5 =		
		Average	X 7 =		
		High	X 10 =		
EIs		Low	X 3 =		
		Average	X 4 =		
		High	X 6 =		
EOs		Low	X 4 =		
		Average	X 5 =		
		High	X 7 =		
EQs		Low	X 3 =		
		Average	X 4 =		
		High	X 6 =		
	Total Unadjusted Function Point Count				

Value Adjustment Factor (VAF)Calculation Table

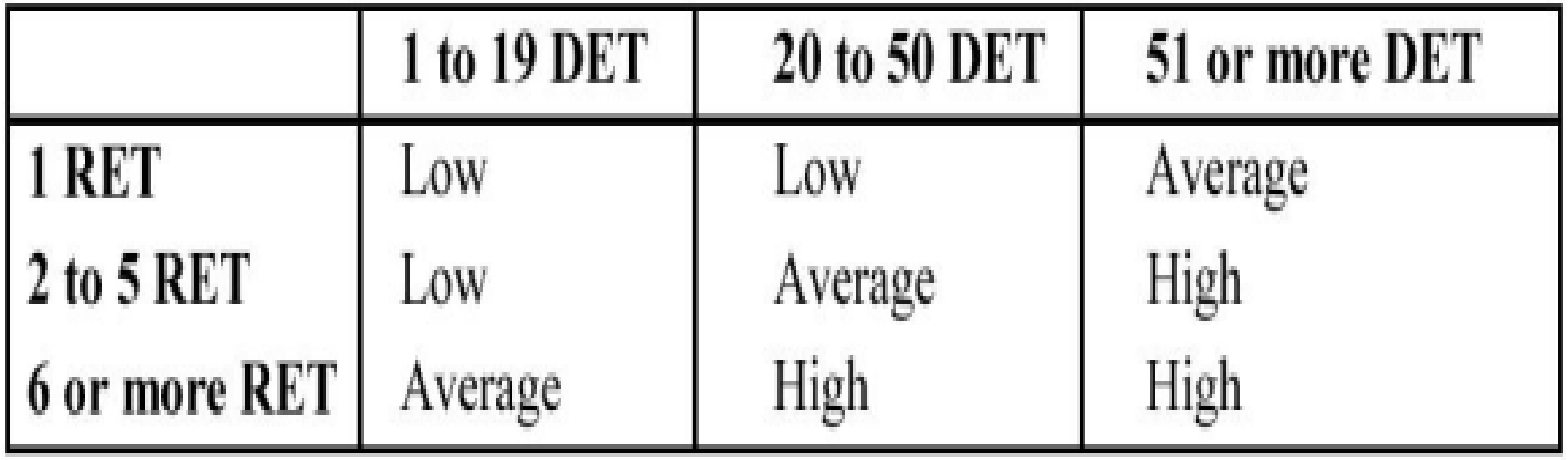
- 0 = No Influence
- 1 = Incidental
- 2 = Moderate
- 3 = Average
- 4 = Significant
- 5 = Essential



General System Characteristics (GSCs)	Degree of Influence (DI) 0 - 5
1. Data Communications	________
2. Distributed Data Processing	________
3. Performance	________
4. Heavily Used Configuration	________
5. Transaction Rate	________
6. Online Data Entry	________
7. End-User Efficiency	________
8. Online Update	________
9. Complex Processing	________
10. Reusability	________
11. Installation Ease	________
12. Operational Ease	________
13. Multiple Sites	________
14. Facilitate Change	________
Total Degree of Influence (TDI)	________
Value Adjustment Factor (VAF)	________

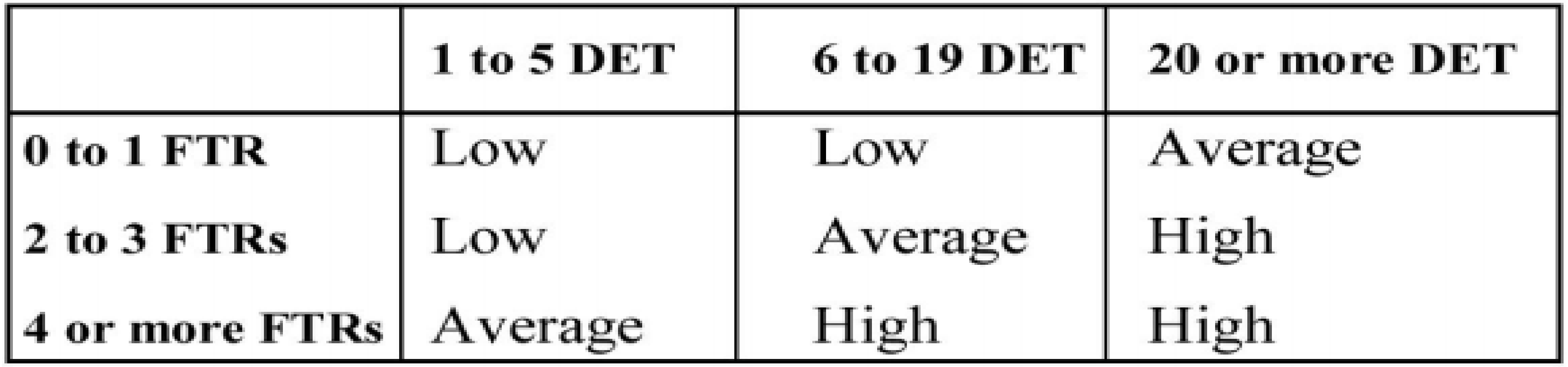
VAF = (TDI * 0.01) + 0.65

Matrix Used for ILF and ELF



	1 to 19 DET	20 to 50 DET	51 or more DET
1 RET	Low	Low	Average
2 to 5 RET	Low	Average	High
6 or more RET	Average	High	High

Matrix Used for EO and EQ



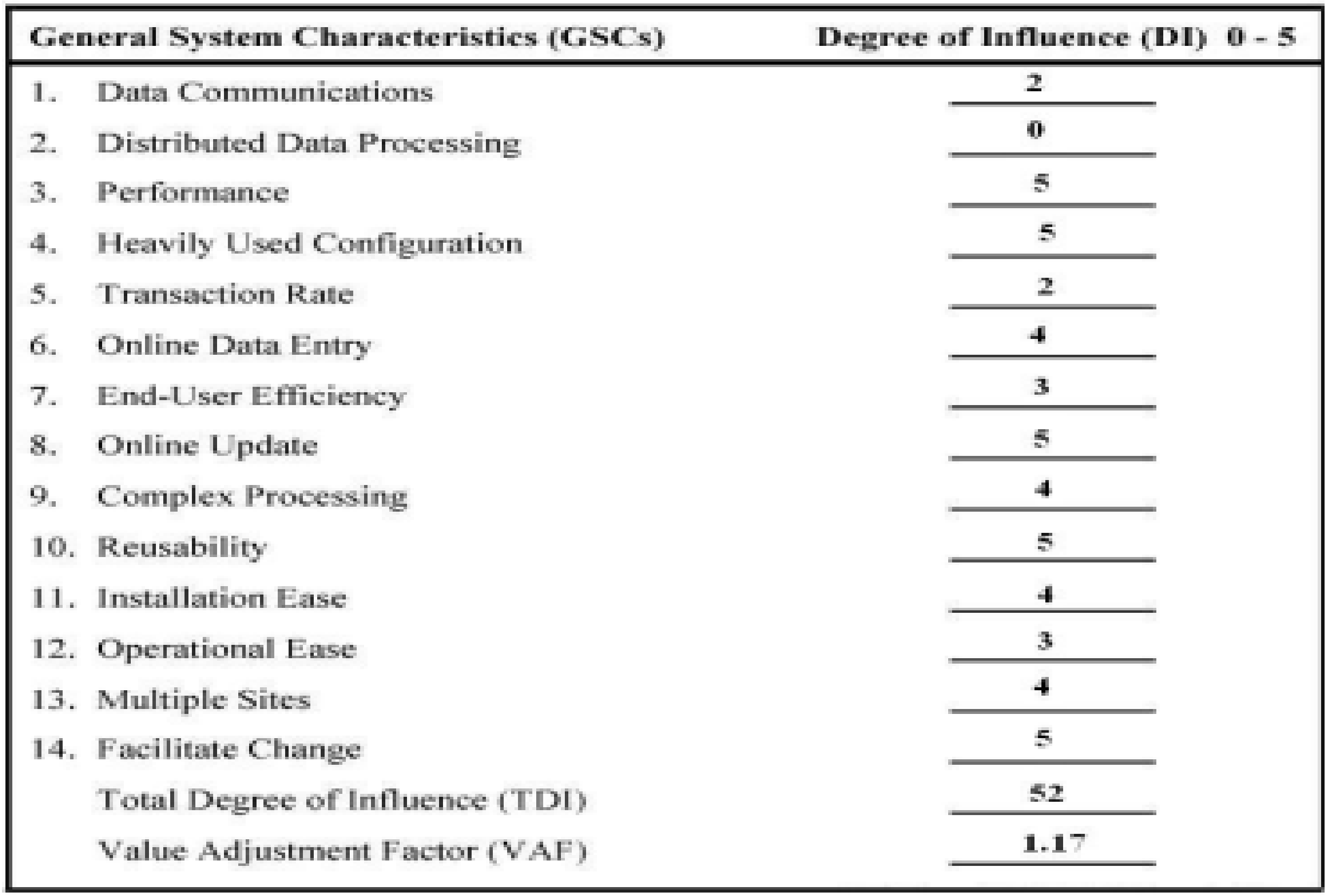
	1 to 5 DET	6 to 19 DET	20 or more DET
0 to 1 FTR	Low	Low	Average
2 to 3 FTRs	Low	Average	High
4 or more FTRs	Average	High	High

An Example

Function Type	Estimated Count	Weight	FP-Count
EI	24	(Average) 4	96
EO	16	(Average) 5	80
EQ	22	(Average) 4	88
ILF	4	(Average) 10	40
ELF	2	(Average) 7	14
UFP count			**318**

Example Continued….

$$VAF = 52 * 0.01 + 0.65$$
$$= 1.17$$
$$FP_{estimated} = 318 \times 1.17$$
$$= 372$$



General System Characteristics (GSCs)	Degree of Influence (DI) 0 - 5
1. Data Communications	2
2. Distributed Data Processing	0
3. Performance	5
4. Heavily Used Configuration	5
5. Transaction Rate	2
6. Online Data Entry	4
7. End-User Efficiency	3
8. Online Update	5
9. Complex Processing	4
10. Reusability	5
11. Installation Ease	4
12. Operational Ease	3
13. Multiple Sites	4
14. Facilitate Change	5
Total Degree of Influence (TDI)	52
Value Adjustment Factor (VAF)	1.17

Feature Point

- Is a superset of FP

Suitable for real-time, process-control and embedded software applications tend to have high algorithmic complexity

Based on the FP measure of software many other metrics can be computed:

1. Errors/FP
2. $/FP.
3. Defects/FP
4. Pages of documentation/FP
5. Errors/PM.
6. Productivity = FP/PM (effort is measured in person-months).
7. $/Page of Documentation.

Another Example

Example: Compute the function point, productivity, documentation, cost per function for the following data:

1. Number of user inputs = 24
2. Number of user outputs = 46
3. Number of inquiries = 8
4. Number of files = 4
5. Number of external interfaces = 2
6. Effort = 36.9 p-m
7. Technical documents = 265 pages
8. User documents = 122 pages
9. Cost = $7744/ month

Various processing complexity factors are: 4, 1, 0, 3, 3, 5, 4, 4, 3, 3, 2, 2, 4, 5.

Solution

Measurement Parameter	Count		Weighing factor
1. Number of external inputs (EI)	24	*	4 = 96
2. Number of external outputs (EO)	46	*	4 = 184
3. Number of external inquiries (EQ)	8	*	6 = 48
4. Number of internal files (ILF)	4	*	10 = 40
5. Number of external interfaces (EIF) Count-total →	2	*	5 = 10 378

So sum of all fi (i ← 1 to 14) = 4 + 1 + 0 + 3 + 5 + 4 + 4 + 3 + 3 + 2 + 2 + 4 + 5 = 43

FP = Count-total * [0.65 + 0.01 *∑(fi)]
= 378 * [0.65 + 0.01 * 43]
= 378 * [0.65 + 0.43]
= 378 * 1.08 = 408

$$\text{Productivity} = \frac{\text{FP}}{\text{Effort}} = \frac{408}{36.9} = 11.1$$

total pages of documentation = technical document + user document = 265 + 122 = 387pages

Documentation = Pages of documentation/FP = 387/408 = 0.94

$$\text{Cost per function} = \frac{\text{cost}}{\text{productivity}} = \frac{7744}{11.1} = \$700$$

THANK YOU

Objectives of Functional Point Analysis

From: https://www.geeksforgeeks.org/software-engineering-functional-point-fp-analysis/ 22/04/2024

1. **Encourage Approximation:** FPA helps in the estimation of the work, time, and materials needed to develop a software project. Organizations can plan and manage projects more accurately when a common measure of functionality is available.

2. **To assist with project management:** Project managers can monitor and manage software development projects with the help of FPA. Managers can evaluate productivity, monitor progress, and make well-informed decisions about resource allocation and project timeframes by measuring the software's functional points.

3. **Comparative analysis:** By enabling benchmarking, it gives businesses the ability to assess how their software projects measure up to industry standards or best practices in terms of size and complexity. This can be useful for determining where improvements might be made and for evaluating how well development procedures are working.

4. • **Improve Your Cost-Benefit Analysis:** It offers a foundation for assessing the value provided by the program concerning its size and complexity, which helps with cost-benefit analysis. Making educated judgements about project investments and resource allocations can benefit from having access to this information.
5. • **Comply with Business Objectives:** It assists in coordinating software development activities with an organization's business objectives. It guarantees that software development efforts are directed toward providing value to end users by concentrating on user-oriented functionality.

Benefits of Functional Point Analysis

1. **Technological Independence:** It calculates a software system's functional size independent of the underlying technology or programming language used to implement it. As a result, it is a technology-neutral metric that makes it easier to compare projects created with various technologies.

2. **Better Accurate Project Estimation:** It helps to improve project estimation accuracy by measuring user interactions and functional needs. Project managers can improve planning and budgeting by using the results of the FPA to estimate the time, effort and resources required for development.

3. **Improved Interaction:** It provides a common language for business analysts, developers, and project managers to communicate with one another and with other stakeholders. By communicating the size and complexity of software in a way that both

technical and non-technical audiences can easily understand this helps close the communication gap.

4. • **Making Well-Informed Decisions:** FPA assists in making well-informed decisions at every stage of the software development life cycle. Based on the functional requirements, organizations can use the results of the FPA to make decisions about resource allocation, project prioritization, and technology selection.
5. • **Early Recognition of Changes in Scope**: Early detection of changes in project scope is made easier with the help of FPA. Better scope change management is made possible by the measurement of functional requirements, which makes it possible to evaluate additions or changes for their effect on the project's overall size.

Characteristics of Functional Point Analysis

We **calculate the functional point** with the help of the number of functions and types of functions used in applications. These are classified into five types:

Measurement Parameters	**Examples**
Number of External Inputs (EI)	Input screen and tables
Number of External Output (EO)	Output screens and reports
Number of external inquiries (EQ)	Prompts and interrupts
Number of internal files (ILF)	Databases and directories
Number of external interfaces (EIF)	Shared databases and shared routines

Functional Point helps in describing system complexity and also shows project timelines. It is majorly used for business systems like information systems.

Weights of 5 Functional Point Attributes

Measurement Parameter	**Low**	**Average**	**High**
Number of external inputs (EI)	3	4	6
Number of external outputs (EO)	4	5	7
Number of external inquiries (EQ)	3	4	6

Measurement Parameter	**Low**	**Average**	**High**
Number of internal files (ILF)	7	10	15
Number of External Interfaces (EIF)	5	7	10

Functional Complexities help us in finding the corresponding weights, which results in finding the Unadjusted Functional point (UFp) of the Subsystem. Consider the complexity as average for all cases. Below-mentioned is the way how to compute FP.

Measurement Parameter	**Count**	Total_Count	**Weighing Factor**		
			Simple	**Average**	**Complex**
Number of external inputs (EI)	32	32*4=128	3	4	6
Number of external outputs (EO)	60	60*5=300	4	5	7
Number of external inquiries (EQ)	24	24*4=96	3	4	6
Number of internal files (ILF)	8	8*10=80	7	10	15
Number of external interfaces (EIF)	2	2*7=14	5	7	10
Algorithms used Count total →		618			

From the above tables, Functional Point is calculated with the following formula

FP = Count-Total * [0.65 + 0.01 * ∑**(fi)**]

= Count * CAF

Here, the **count-total** is taken from the chart.

CAF = [0.65 + 0.01 * ∑**(fi)**]

1. ∑**(fi) =** sum of all 14 questions and it also shows the complexity factor – CAF.

2. CAF varies from 0.65 to 1.35 and ∑**(fi)** ranges from 0 to 70.

3. When ∑**(fi)** = 0, CAF = 0.65 and when ∑**(fi)** = 70, CAF = 0.65 + (0.01*70) = 0.65 + 0.7 = 1.35

Questions on Functional Point??

= = ================================

Software Engineering | Calculation of Function Point (FP)

Last Updated : 28 Jun, 2020 from: https://www.geeksforgeeks.org/software-engineering-calculation-of-function-point-fp/ 22/04/2024

-
-
-

Function Point (FP) is an element of software development which helps to approximate the cost of development early in the process. It may measures functionality from user's point of view.

Counting Function Point (FP):

- **Step-1:**

```
F = 14 * scale
```

- Scale varies from 0 to 5 according to character of Complexity Adjustment Factor (CAF). Below table shows scale:

```
0 - No Influence
1 - Incidental
2 - Moderate
3 - Average
4 - Significant
5 - Essential
```

- **Step-2:** Calculate Complexity Adjustment Factor (CAF).

```
CAF = 0.65 + ( 0.01 * F )
```

- **Step-3:** Calculate Unadjusted Function Point (UFP).

TABLE (Required)

Function Units	Low	Avg	High
EI	3	4	6
EO	4	5	7
EQ	3	4	6
ILF	7	10	15

Function Units	Low	Avg	High
EIF	5	7	10

Multiply each individual function point to corresponding values in TABLE.

- **Step-4:** Calculate Function Point.

```
FP = UFP * CAF
```

Example:
Given the following values, compute function point when all complexity adjustment factor (CAF) and weighting factors are average.

```
User Input = 50
User Output = 40
User Inquiries = 35
User Files = 6
External Interface = 4
```

Explanation:

- **Step-1:** As complexity adjustment factor is average (given in question), hence,
- `scale = 3.`

```
F = 14 * 3 = 42
```

- **Step-2:**

```
CAF = 0.65 + ( 0.01 * 42 ) = 1.07
```

- **Step-3:** As weighting factors are also average (given in question) hence we will multiply each individual function point to corresponding values in TABLE.

```
UFP = (50*4) + (40*5) + (35*4) + (6*10) + (4*7) = 628
```

- **Step-4:**

```
Function Point = 628 * 1.07 = 671.96
```

This is the required answer.

SWE 402 Lecture Note Draft3d More Materials on FPA

WHAT IS FUNCTIONAL POINT ANALYSIS WITH AN EXAMPLE?

Function Point Analysis (FPA) is a standardized method to measure the functional size of a software application. It quantifies the functionality provided to the user based on the logical design and requirements of the system, rather than the technical or physical aspects.

### **Key Concepts in Function Point Analysis**

1. **Functional User Requirements (FUR)**: These are the functional requirements that describe what the system does. FPA focuses on these requirements.

2. **Unadjusted Function Points (UFP)**: The raw count of function points based on the identified functions and their complexities.

3. **Value Adjustment Factor (VAF)**: A factor that adjusts the UFP to account for various system characteristics that influence productivity.

4. **Function Types**:

   - **External Inputs (EI)**: Inputs entering the system from the outside (e.g., forms, data entry screens).

   - **External Outputs (EO)**: Outputs leaving the system (e.g., reports, messages).

   - **External Inquiries (EQ)**: Interactive inputs requiring an immediate response (e.g., query screens).

   - **Internal Logical Files (ILF)**: User-identifiable groups of related data within the system (e.g., databases).

   - **External Interface Files (EIF)**: User-identifiable groups of related data used for reference purposes that reside outside the system (e.g., shared databases).

### Steps in Function Point Analysis

1. **Identify and classify functions**:

   - Identify all EIs, EOs, EQs, ILFs, and EIFs.

- Classify each based on complexity (Low, Average, High).

2. **Calculate Unadjusted Function Points (UFP)**:

   - Assign weights to each function type based on complexity using a predefined table.

   - Sum the weights to get the UFP.

3. **Determine the Value Adjustment Factor (VAF)**:

   - Evaluate 14 general system characteristics (GSCs) such as data communications, performance, and transaction rate.

   - Each characteristic is rated on a scale from 0 to 5.

   - Calculate the VAF using the formula: VAF = 0.65 + (0.01 * Sum of GSC ratings).

4. **Calculate the Adjusted Function Points (AFP)**:

   - Multiply the UFP by the VAF to get the AFP: AFP = UFP * VAF.

### Example of Function Point Analysis

Let's analyze a simple library management system:

1. **Identify and classify functions**:

   - **External Inputs (EI)**:

     - Add new book (Average)

     - Update book information (Average)

   - **External Outputs (EO)**:

     - Generate overdue report (Low)

   - **External Inquiries (EQ)**:

     - Search book (Average)

   - **Internal Logical Files (ILF)**:

     - Books database (High)

- Members database (Average)

- **External Interface Files (EIF)**:

  - External publisher database (Low)

2. **Calculate Unadjusted Function Points (UFP)**:

   - EIs: 2 (Average) * 4 (weight for Average EI) = 8

   - EOs: 1 (Low) * 4 (weight for Low EO) = 4

   - EQs: 1 (Average) * 4 (weight for Average EQ) = 4

   - ILFs: 1 (High) * 10 (weight for High ILF) + 1 (Average) * 7 (weight for Average ILF) = 10 + 7 = 17

   - EIFs: 1 (Low) * 5 (weight for Low EIF) = 5

   Total UFP = 8 (EIs) + 4 (EOs) + 4 (EQs) + 17 (ILFs) + 5 (EIFs) = 38

3. **Determine the Value Adjustment Factor (VAF)**:

   - Assume the sum of GSC ratings is 30.

   - VAF = 0.65 + (0.01 * 30) = 0.65 + 0.3 = 0.95

4. **Calculate the Adjusted Function Points (AFP)**:

   - AFP = UFP * VAF = 38 * 0.95 = 36.1

Thus, the adjusted function points for the library management system are 36.1.

Function Point Analysis helps in estimating the effort required for development, assessing productivity, and comparing different software projects based on their functional size.

= =

Uses of function point analysis

Function Point Analysis (FPA) is a versatile tool used in various aspects of software development and management. Here are some of the primary uses:

### 1. **Effort Estimation and Project Planning**

FPA provides a way to estimate the effort required to develop or enhance a software application. By understanding the functional size of a project, project managers can make more accurate predictions about the resources, time, and budget needed.

### 2. **Cost Estimation**

The functional size measured in function points can be used to estimate the cost of a project. This is particularly useful in budgeting and financial planning, as function points can be converted into monetary terms based on historical data on cost per function point.

### 3. **Productivity Measurement**

Function points enable the measurement of productivity by comparing the number of function points delivered to the amount of effort expended (e.g., person-hours or person-days). This helps in benchmarking productivity across projects and teams.

### 4. **Performance Benchmarking**

Organizations can use function points to benchmark their performance against industry standards or competitors. By comparing function point metrics, they can identify areas for improvement and set realistic performance goals.

### 5. **Quality Assurance**

Function points can be correlated with defect rates to assess the quality of software. Higher function point counts might be associated with higher defect counts, helping in identifying the relationship between system complexity and quality.

### 6. **Scope Management**

FPA helps in managing scope changes by quantifying the impact of adding, modifying, or deleting functionalities. This ensures that stakeholders understand the implications of scope changes on project effort, time, and cost.

### 7. **Contract Management**

In outsourcing and vendor management, function points provide a standardized way to define and measure the deliverables. This helps in creating clear, quantifiable contracts and service level agreements (SLAs).

### 8. **Maintenance and Enhancement**

For ongoing maintenance and enhancement projects, function points help in estimating the effort required to add new features or modify existing ones. This assists in resource allocation and scheduling for maintenance activities.

### 9. **Portfolio Management**

Organizations can use FPA to evaluate and prioritize projects within their portfolio. By comparing the functional sizes and associated costs, they can make informed decisions about which projects to pursue, defer, or cancel.

### 10. **Historical Data Analysis**

Function points provide a basis for analyzing historical project data. Organizations can identify trends, improve estimation models, and refine processes based on lessons learned from past projects.

### 11. **Risk Management**

By quantifying the functional size, FPA helps in identifying potential risks related to project complexity. Larger function point counts may indicate higher complexity and thus higher risk, allowing for proactive risk mitigation strategies.

### 12. **User Communication**

FPA offers a way to communicate project scope and complexity to non-technical stakeholders in a clear and understandable manner. This helps in setting expectations and ensuring alignment between the development team and stakeholders.

Overall, Function Point Analysis is a robust method for quantifying and managing various aspects of software development, providing a standardized framework for measuring and improving software project performance.

Software Engineering Economics
COCOMO Model
Basic COCOMO
Intermediate COCOMO

(Slightly modified by E.K.Olatunji, TAU, Oko on 27-05-2024)

Instructor Name: Riaz Ahmad

Department of Computer Science, COMSATS University Islamabad, Wah Campus, Pakistan

Outline

- What is COCOMO Model
- Introduction of The Topic
- COCOMO, projects are categorized into three types:
- How it works(How to do)?
- Intermediate Model:
- Summary of the lecture
- What next……..

Bisconni
Constructive Cost Estimation Model
Cocomo
~~CHOCOLATE FILLED BISCUITS~~

What is COCOMO Model?

Boehm proposed COCOMO (Constructive Cost Estimation Model) in 1981.COCOMO is one of the most generally used software estimation models in the world. COCOMO predicts the efforts and schedule of a software product based on the size of the software.

The necessary steps in this model are:

1.Get an initial estimate of the development effort from evaluation of thousands of delivered lines of source code (KDLOC).

2.Determine a set of 15 multiplying factors from various attributes of the project.

3.Calculate the effort estimate by multiplying the initial estimate with all the multiplying factors i.e., multiply the values in step1 and step2.

Introduction of The Topic

The initial estimate (also called nominal estimate) is determined by an equation of the form used in the static single variable models, using KDLOC as the measure of the size. To determine the initial effort Ei in person-months the equation used is of the type is shown below

$$Ei=a*(KDLOC) ^ b$$

COCOMO, projects are categorized into three types:

The values of the constant a and b are dependent on the project type.

In COCOMO, projects are categorized into three types:

1. Organic
2. Semidetached
3. Embedded

Organic: A development project can be treated as the organic type, if the project deals with developing a well-understood application program, the size of the development team is reasonably small, and the team members are experienced in developing similar methods of projects. Examples of this type of project are simple business systems, simple inventory management systems, and data processing systems.

COCOMO, projects are categorized into three types:

Semidetached: A development project can be treated with semidetached type if the development consists of a mixture of experienced and inexperienced staff. Team members may have finite experience in related systems but may be unfamiliar with some aspects of the other being developed. Example of a Semidetached system includes developing a new operating system (OS), a Database Management System (DBMS), and a complex inventory management system.

Embedded: A development project is treated to be of an embedded type, if the software being developed is strongly coupled to complex hardware, or if the stringent regulations on the operational method exist.

Summary of Project Attributes

- **Organic:**

  2-50 KLOC small, stable, little innovation

- **Semi-Detached:**

  50-300 KLOC medium-sized, average abilities, medium time-constraints

- **Embedded:**

  > 300 KLOC large project team, complex, innovative, severe constraints

An Example

For Example: ATM, Air Traffic control.

For three product categories, Bohem provides a different set of expression to predict effort (in a unit of person month)and development time from the size of estimation in KLOC(Kilo Line of code) efforts estimation considers the productivity loss due to holidays, weekly off, coffee breaks, etc.

According to Boehm, software cost estimation should be done through three stages:

1. Basic Model
2. Intermediate Model
3. Detailed Model

How it works(How to do)?

Basic COCOMO Model: The basic COCOMO model provide an accurate size of the project parameters. The following expressions give the basic COCOMO estimation model:

Effort=a1*(KLOC) ^ a2 PM

Tdev=b1*(efforts) ^ b2 Months

Personnel required = Effort/Tdev

Where **KLOC** is the estimated size of the software product indicated in Kilo Lines of Code, a1,a2,b1,b2 are constants for each group of software products,
Tdev is the estimated time to develop the software, expressed in months, **Effort** is the total effort required to develop the software product, expressed in person months (PMs).

How it works(How to do)?

Estimation of development effort

For the three classes of software products, the formulas for estimating the effort based on the code size are shown below:

Organic: Effort = 2.4(KLOC) 1.05 PM (actually 2.4*(KLOC) ^ 1.05)

Semi-detached: Effort = 3.0(KLOC) 1.12 PM (actually 3.0 * (KLOC) ^ 1.12)

Embedded: Effort = 3.6(KLOC) 1.20 PM (actually 3.6 * (KLOC) ^ 1.20)

Estimation of development time

For the three classes of software products, the formulas for estimating the development time based on the effort are given below:

Organic: Tdev = 2.5(Effort) 0.38 Months (Actually 2.5*(Effort) ^ 0.38)

Semi-detached: Tdev = 2.5(Effort) 0.35 Months (Actually 2.5 (Effort) ^ 0.35)

Embedded: Tdev = 2.5(Effort) 0.32 Months (actually 2.5 * (Effort) ^ 0.32)

A little Bit More…..

The **basic COCOMO model** can be obtained by plotting the estimated characteristics for different software sizes.

Fig-1 shows a plot of estimated effort versus product size.

Fig-2 we can observe that the effort is somewhat superliner in the size of the software product. Thus, the effort required to develop a product increases very rapidly with project size.

Effort Vs. Product Size

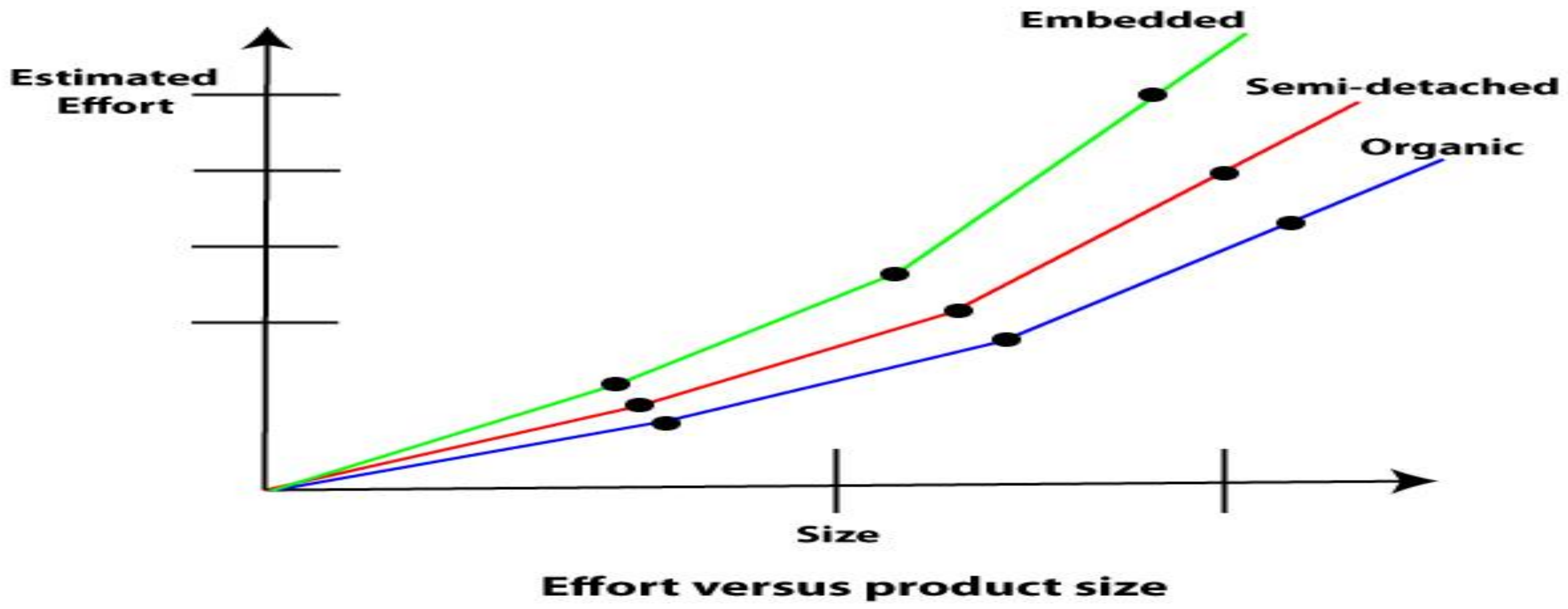


Fig-1

Development Time Vs. Size

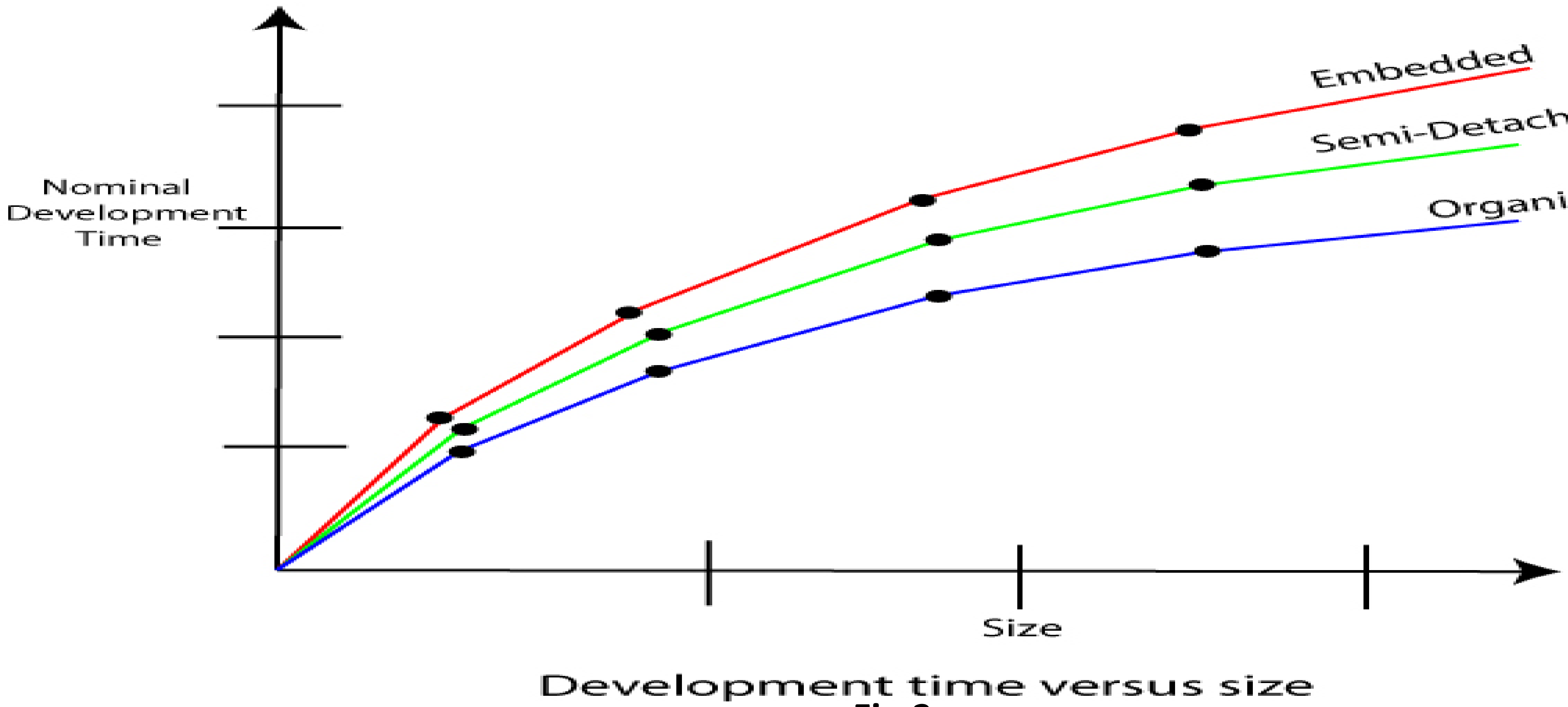


Fig-2

Explanation……

The **development time** versus the **product size in KLOC** is plotted in fig-2. it can be observed that , i.e. when the size of the product increases by two times, the time to develop the product does not double but rises moderately. This can be explained by the fact that for larger products, a larger number of activities which can be carried out concurrently can be identified. The parallel activities can be carried out simultaneously by the engineers. This reduces the time to complete the project. Further, from fig, it can be observed that the development time is roughly the same for all three categories of products. For example, a 60 KLOC program can be developed in approximately 18 months, regardless of whether it is of organic, semidetached, or embedded type.

Other Estimates Can Be Obtained….

From the effort estimation, the **project cost** can be obtained by multiplying the required effort by the manpower cost per month. But, implicit in this project cost computation is the assumption that the entire project cost is incurred on account of the manpower cost alone. In addition to manpower cost, a project would incur costs due to hardware and software required for the project and the company overheads for administration, office space, etc.

It is important to note that the effort and the duration estimations obtained using the COCOMO model are called a **nominal effort** estimate and **nominal duration** estimate. The term nominal implies that if anyone tries to complete the project in a time shorter than the estimated duration, then the cost will increase drastically. But, if anyone completes the project over a longer period than the estimated, then there is almost no decrease in the estimated cost value.

Example1:

Example1: Suppose a project was estimated to be 400 KLOC. Calculate the effort and development time for each of the three model i.e., organic, semi-detached & embedded.

Solution: The basic COCOMO equation takes the form:

Effort=a1*(KLOC) ^ a2 PM
Tdev=b1*(efforts) ^ b2 Months
Estimated Size of project= 400 KLOC

(i)Organic Mode

E = 2.4 * (400) ^ 1.05 = 1295.31 PM
D = 2.5 * (1295.31) ^ 0.38=38.07 M

(ii)Semidetached Mode

E = 3.0 * (400) ^ 1.12=2462.79 PM
D = 2.5 * (2462.79) ^0.35=38.45 M

(iii) Embedded Mode

E = 3.6 * (400)^ 1.20 = 4772.81 PM
D = 2.5 * (4772.8)^ 0.32 = 38 M

Example2:

Example2: A project size of 200 KLOC is to be developed. Software development team has average experience on similar type of projects. The project schedule is not very tight. Calculate the Effort, development time, average staff size, and productivity of the project.

Solution: The semidetached mode is the most appropriate mode, keeping in view the size, schedule and experience of development time.

E=3.0(200) ^ 1.12=1133.12PM

D=2.5(1133.12) ^ 0.35=29.3PM

P = 176 LOC/PM

$$\text{Average Staff Size (SS)} = \frac{E}{D} \text{ Persons}$$

$$= \frac{1133.12}{29.3} = 38.67 \text{ Persons}$$

$$\text{Productivity} = \frac{\text{KLOC}}{E} = \frac{200}{1133.12} = 0.1765 \text{ KLOC/PM}$$

Intermediate Model:

The cost drivers are divided into four categories:

Intermediate Model: Cost Drivers

Intermediate Model: The basic Cocomo model considers that the effort is only a function of the number of lines of code and some constants calculated according to the various software systems. The intermediate COCOMO model recognizes these facts and refines the initial estimates obtained through the basic COCOMO model by using a set of 15 cost drivers based on various attributes of software engineering.

Classification of Cost Drivers and their attributes:

Product attributes -

- Required software reliability extent
- Size of the application database
- The complexity of the product

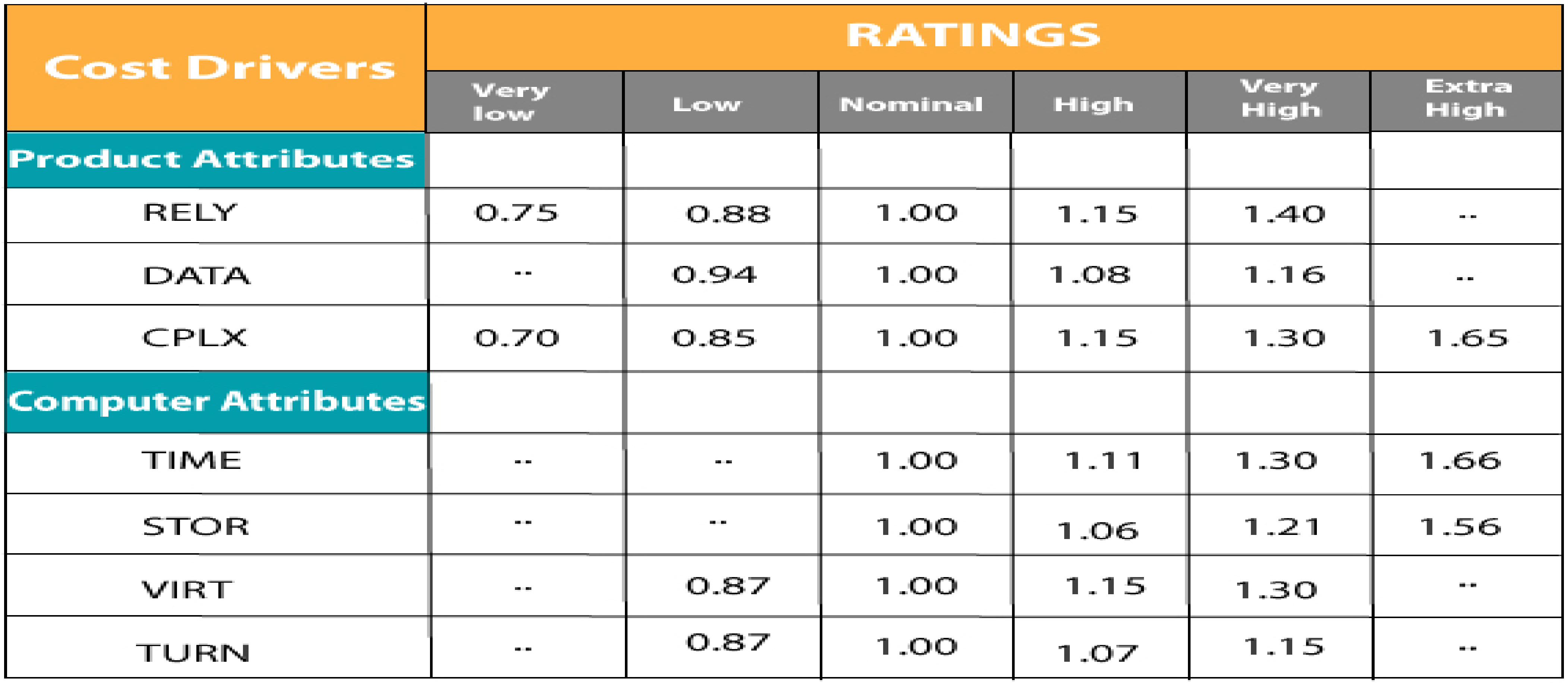
Hardware attributes -

- Run-time performance constraints
- Memory constraints
- The volatility of the virtual machine environment
- Required turnabout time

Intermediate Model: Cost Drivers

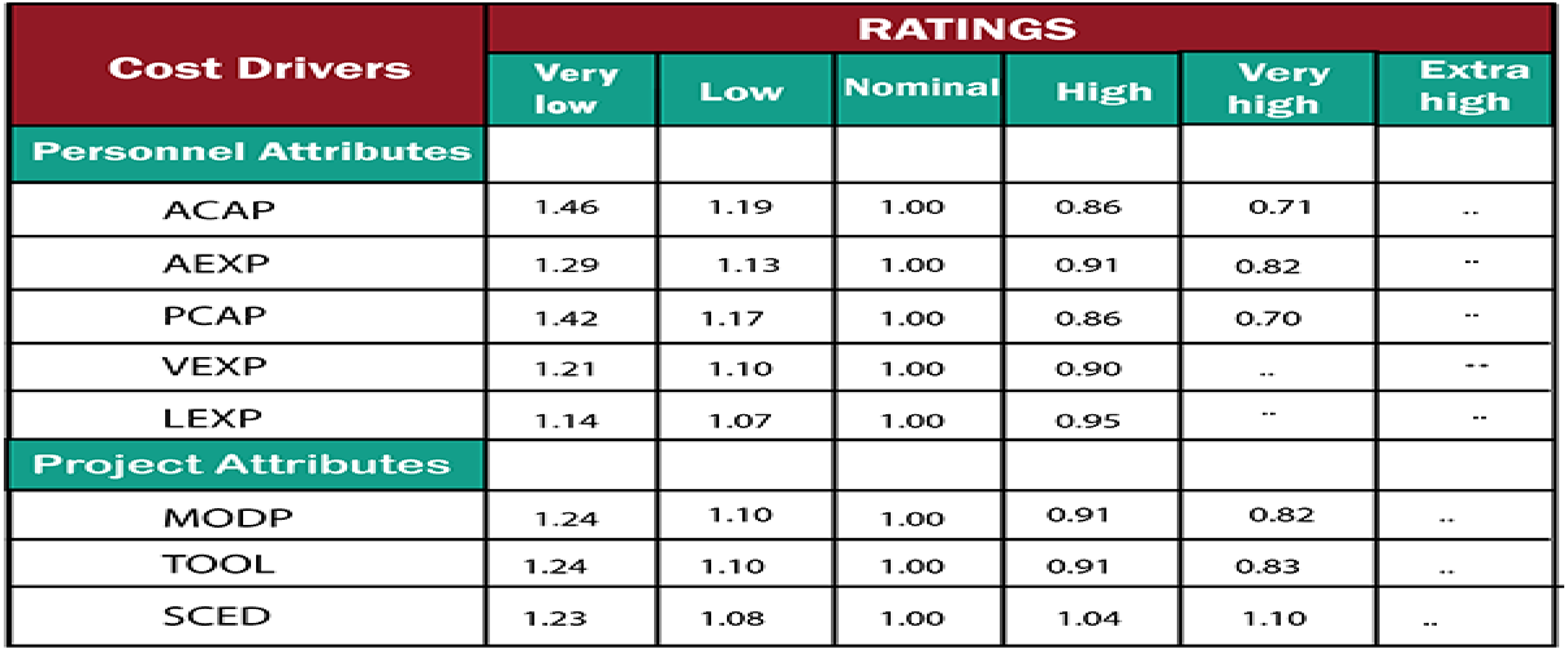
- **Personnel attributes**
  - analyst capability
  - software engineer capability
  - applications experience
  - virtual machine experience
  - programming language experience
- **Project attributes**
  - use of software tools
  - application of software engineering methods
  - required development schedule

Cost Drivers of Intermediate COCOMO Model



Cost Drivers	RATINGS					
	Very low	Low	Nominal	High	Very High	Extra High
Product Attributes						
RELY	0.75	0.88	1.00	1.15	1.40	--
DATA	--	0.94	1.00	1.08	1.16	--
CPLX	0.70	0.85	1.00	1.15	1.30	1.65
Computer Attributes						
TIME	--	--	1.00	1.11	1.30	1.66
STOR	--	--	1.00	1.06	1.21	1.56
VIRT	--	0.87	1.00	1.15	1.30	--
TURN	--	0.87	1.00	1.07	1.15	--

Cost Drivers of Intermediate COCOMO Model



Cost Drivers	RATINGS					
	Very low	Low	Nominal	High	Very high	Extra high
Personnel Attributes						
ACAP	1.46	1.19	1.00	0.86	0.71	..
AEXP	1.29	1.13	1.00	0.91	0.82	--
PCAP	1.42	1.17	1.00	0.86	0.70	--
VEXP	1.21	1.10	1.00	0.90	..	--
LEXP	1.14	1.07	1.00	0.95	--	--
Project Attributes						
MODP	1.24	1.10	1.00	0.91	0.82	..
TOOL	1.24	1.10	1.00	0.91	0.83	..
SCED	1.23	1.08	1.00	1.04	1.10	..

Intermediate COCOMO equation:

Intermediate COCOMO equation:

E=ai * (KLOC) ^ bi*EAF (EAF = Effort Adjusted Factors)

Solution:

3.6 * (355) ^ 1.20 *(1.40*1.16*1.30*1.21*1.30*1.15*0.86*1.10*0.95*1.91*1.91*1.91*1.09)

= 4135.96*15.49

=64066.02 PM

D=ci * (E) ^ di

=2.5 * (64066.02) ^ 0.32

=86.30 M

<u>Coefficients for intermediate COCOMO</u>

Project	ai	bi	ci	di
Organic	2.4	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

A Problem (Intermediate COCOMO)

Problem:

Suppose you are working in a team developing a real time application to observe the smog level, duration and timing on various points of motorways. The system will be helpful to manage the traffic and avoid the number of accidents on the motorway. Suppose that the size of the project is 355KLOC. Customer (motorway police) expects highly reliable application. The size of database is comparatively bigger because of graphical data. Various components of application communicate with each other. The application required high computational time, large memory to store the data, virtual machine for visualization and average turn around time. The development team is highly motivated containing new members recently graduated from the university with good programming skills with no virtual machine experience and less domain knowledge. Motorway police expects that developers should use best available language and tools within schedule time. You are required to calculate effort and development time for this project.

Solution:

Suppose you are working in a team developing a real time application to observe the smog level, duration and timing on various points of motorways. The system will be helpful to manage the traffic and avoid the number of accidents on the motorway.

A Problem (Intermediate COCOMO)

1) Product attributes

Customer (motorway police) expects highly reliable application. 1.40

The size of database is comparatively bigger because of graphical data. Various components of application communicate with each other. 1.16

2) Computer Attributes

The application required high computational time, large memory to store the data, virtual machine for visualization and average turn around time. 1.30,1.21,1.30,1.15

3) Personal Attributes

The development team is highly motivated containing new members recently graduated from the university with good programming skills with no virtual machine experience and less domain knowledge.0.86,1.10,0.95,1.19

4) Project Attributes

Motorway police expects that developers should use best available language and tools within schedule time. You are required to calculate effort and development time for this project.1.91,19.1.1.04

A Problem (Intermediate COCOMO)

Size of project = 355 KLOC

Identify Cost drivers

1) Product attributes

Reliability =V. high

Data base size = V. high

Project Complexity(Components communication) =Extra high

2) Computer Attributes

Execution time = V. high

Constraint = Extra high

Virtual Machine Volatility = High

Turn Around time = avg

A Problem (Intermediate COCOMO)

3) Personal Attributes

Analyst Capability = Low

Application Experience = Very low

Programmer Capability = V. High

Virtual machine Experience = Very Low

Programming Language Experience = Normal

4) Project Attributes

Modern programming Practices = V. High

Use of SW tool = V. High

Required Dev. Schedule = V. High

Solution:

Intermediate COCOMO equation:

E=ai * (KLOC) ^ bi*EAF

Solution:

3.6 * (355)^1.20 *(1.40*1.16*1.30*1.21*1.30*1.15*0.86*1.10*0.95*1.91*1.91*1.91*1.09)

= 4135.96*15.49

=64066.02 PM

D=ci *(E) ^di

=2.5*(64066.02) ^0.32

=86.30 M

Do it
yourself

THANK YOU

What is the Cocomo Model?

From: https://www.geeksforgeeks.org/software-engineering-cocomo-model/ on 04-04-2024

The Cocomo Model is a procedural cost estimate model for software projects and is often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time, and quality. It was proposed by Barry Boehm in 1981 and is based on the study of 63 projects, which makes it one of the best-documented models.

The key parameters that define the quality of any software products, which are also an outcome of the Cocomo are primarily Effort and schedule:

1. **Effort:** Amount of labor that will be required to complete a task. It is measured in person-months units.

2. **Schedule:** This simply means the amount of time required for the completion of the job, which is, of course, proportional to the effort put in. It is measured in the units of time such as weeks, and months.

1. Organic

A software project is said to be an organic type if the team size required is adequately small, the problem is well understood and has been solved in the past and also the team members have a nominal experience regarding the problem.

2. Semi-detached

A software project is said to be a Semi-detached type if the vital characteristics such as team size, experience, and knowledge of the various programming environments lie in between organic and embedded. The projects classified as Semi-Detached are comparatively less familiar and difficult to develop compared to the organic ones and require more experience better guidance and creativity. Eg: Compilers or different Embedded Systems can be considered Semi-Detached types.

3. Embedded

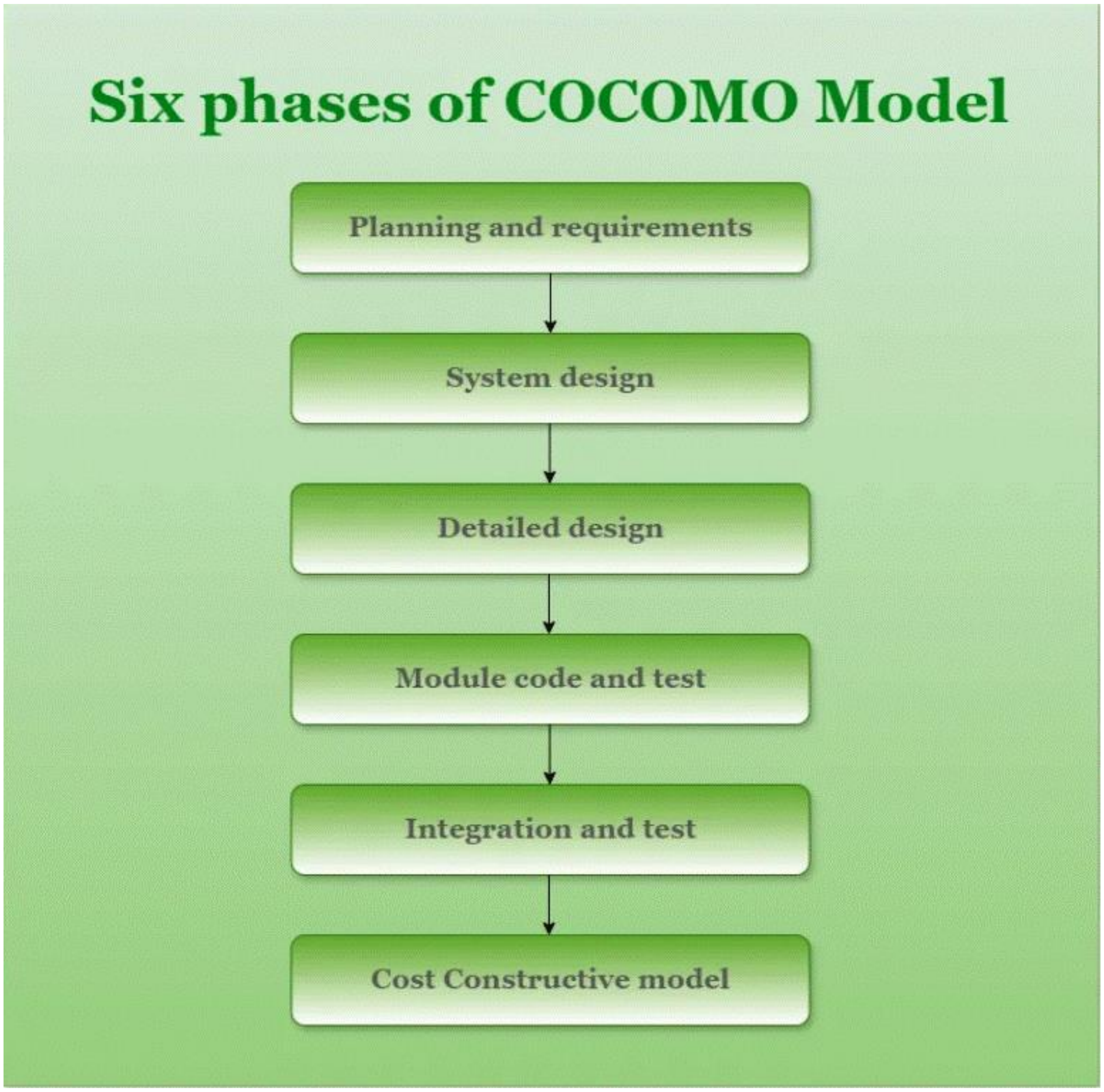
A software project requiring the highest level of complexity, creativity, and experience requirement falls under this category. Such software requires a larger team size than the other two models and also the developers need to be sufficiently experienced and creative to develop such complex models.

Detailed Structure of COCOMO Model

Detailed COCOMO incorporates all characteristics of the intermediate version with an assessment of the cost driver's impact on each step of the software engineering process. The detailed model uses different effort multipliers for each cost driver attribute. In detailed Cocomo, the whole software is divided into different modules and then we apply COCOMO in different modules to estimate effort and then sum the effort.

The Six phases of detailed COCOMO are:

1. **Planning and requirements**
2. **System design**
3. **Detailed design**
4. **Module code and test**
5. **Integration and test**
6. **Cost Constructive model**



Different models of Cocomo have been proposed to predict the cost estimation at different levels, based on the amount of accuracy and correctness required. All of these models can be applied to a variety of projects, whose characteristics determine the value of the constant to be used in subsequent calculations. These characteristics of different system types are mentioned below. Boehm's definition of organic, semidetached, and embedded systems:

Importance of the COCOMO Model

1. **Cost Estimation:** To help with resource planning and project budgeting, COCOMO offers a methodical approach to software development cost estimation.

2. **Resource Management:** By taking team experience, project size, and complexity into account, the model helps with efficient resource allocation.

3. **Project Planning**: COCOMO assists in developing practical project plans that include attainable objectives, due dates, and benchmarks.

4. **Risk management**: Early in the development process, COCOMO assists in identifying and mitigating potential hazards by including risk elements.

5. **Support for Decisions**: During project planning, the model provides a quantitative foundation for choices about scope, priorities, and resource allocation.

6. • **Benchmarking**: To compare and assess various software development projects to industry standards, COCOMO offers a benchmark.
7. • **Resource Optimization:** The model helps to maximize the use of resources, which raises productivity and lowers costs.

Types of COCOMO Model

1. Basic Model

E = a(KLOC)^b

Time = c(Effort)^d

Person required = Effort/ time

The above formula is used for the cost estimation of the basic COCOMO model and also is used in the subsequent models. The constant values a, b, c, and d for the Basic Model for the different the different categories of the system:

Software Projects	**a**	**b**	**c**	**d**
Organic	2.4	1.05	2.5	0.38
Semi-Detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

1. The effort is measured in Person-Months and as evident from the formula is dependent on Kilo-Lines of code. The development time is measured in months.

2. These formulas are used as such in the Basic Model calculations, as not much consideration of different factors such as reliability, and expertise is taken into account, henceforth the estimate is rough.

3. **Below are the programs for Basic COCOMO:**
4. CPP

```
// C++ program to implement basic COCOMO
#include <bits/stdc++.h>

using namespace std;

// Function For rounding off float to int
int fround(float x)
{
    int a;
    x = x + 0.5;
    a = x;
    return (a);
}
```

```
// Function to calculate parameters
// of Basic COCOMO
void calculate(float table[][4], int n,
               char mode[][15], int size)
{
    float effort, time, staff;

    int model;

    // Check the mode according to size
    // organic
    if (size >= 2 && size <= 50)
        model = 0;

    // semi-detached
    else if (size > 50 && size <= 300)
        model = 1;

    // embedded
    else if (size > 300)
        model = 2;

    cout << "The mode is " << mode[model];

    // Calculate Effort
// Calculate Effort
    effort = table[model][0] * pow(size,
                               table[model][1]);

    // Calculate Time
    time = table[model][2] * pow(effort,
                             table[model][3]);

    // Calculate Persons Required
    staff = effort / time;

    // Output the values calculated
    cout << "\nEffort = " << effort <<
            " Person-Month";

    cout << "\nDevelopment Time = " << time <<
            " Months";

    cout << "\nAverage Staff Required = " <<
            fround(staff) << " Persons";
}
// Driver code
int main()
{
    float table[3][4] = {2.4, 1.05, 2.5, 0.38, 3.0, 1.12,
                         2.5, 0.35, 3.6, 1.20, 2.5, 0.32};

    char mode[][15]
        = {"Organic", "Semi-Detached", "Embedded"};
    int size = 4;

    calculate(table, 3, mode, size);

    return 0;
}
```

Java

```
import java.util.Arrays;

public class BasicCOCOMO
{
    private static final double[][] TABLE =
    {
        {2.4, 1.05, 2.5, 0.38},
        {3.0, 1.12, 2.5, 0.35},
        {3.6, 1.20, 2.5, 0.32}
    };
    private static final String[] MODE =
    {
        "Organic", "Semi-Detached", "Embedded"
    };

    public static void calculate(int size)
    {
        int model = 0;

        // Check the mode according to size
        if (size >= 2 && size <= 50)
        {
model = 0;
        } else if (size > 50 && size <= 300)
        {
            model = 1;
        } else if (size > 300)
        {
            model = 2;
        }

        System.out.println("The mode is " + MODE[model]);

        // Calculate Effort
        double effort = TABLE[model][0] * Math.pow(size,
                                              TABLE[model][1]);

        // Calculate Time
        double time = TABLE[model][2] * Math.pow(effort,
                                            TABLE[model][3]);

        // Calculate Persons Required
        double staff = effort / time;
// Output the values calculated
        System.out.println("Effort = " + Math.round(effort) +
                           " Person-Month");
        System.out.println("Development Time = " + Math.round(time) +
                           " Months");
        System.out.println("Average Staff Required = " + Math.round(staff) +

                           " Persons");
    }

    public static void main(String[] args)
    {
        int size = 4;
        calculate(size);
    }
}
```

Output

```
The mode is Organic
Effort = 10.289 Person-Month
Development Time = 6.06237 Months
Average Staff Required = 2 Persons
```

2. Intermediate Model

The basic Cocomo model assumes that the effort is only a function of the number of lines of code and some constants evaluated according to the different software systems. However, in reality, no system's effort and schedule can be solely calculated based on Lines of Code.

For that, various other factors such as reliability, experience, and Capability. These factors are known as **Cost Drivers** and the Intermediate Model utilizes 15 such drivers for cost estimation.

Classification of Cost Drivers and their Attributes:

Product attributes:

1. Required software reliability extent
2. Size of the application database
3. The complexity of the product

Hardware Attributes

4. Run-time performance constraints
5. Memory constraints
6. The volatility of the virtual machine environment
7. Required turnabout time

Personnel Attributes

8. Analyst capability
9. Software engineering capability
10. Application experience
11. Virtual machine experience
12. Programming language experience

Project Attributes

13. Use of software tools

14. Application of software engineering methods

15. Required development schedule

CASE Studies and Examples

1. **NASA Space Shuttle Software Development:** NASA estimated the time and money needed to build the software for the Space Shuttle program using the COCOMO model. NASA was able to make well-informed decisions on resource allocation and project scheduling by taking into account variables including project size, complexity, and team experience.

2. **Big Business Software Development:** The COCOMO model has been widely used by big businesses to project the time and money needed to construct intricate business software systems. These organizations were able to better plan and allocate resources for their software projects by using COCOMO's estimation methodology.

3. **Commercial Software goods:** The COCOMO methodology has proven advantageous for software firms that create commercial goods as well. These businesses were able to decide on pricing, time-to-market, and resource allocation by precisely calculating the time and expense of building new software products or features.

4. **Academic Research Initiatives:** To estimate the time and expense required to create software prototypes or carry out experimental studies, academic research initiatives have employed COCOMO. Researchers were able to better plan their projects and allocate resources by using COCOMO's estimate approaches.

Advantages of the COCOMO Model

1. **Systematic cost estimation:** Provides a systematic way to estimate the cost and effort of a software project.

2. **Helps to estimate cost and effort:** This can be used to estimate the cost and effort of a software project at different stages of the development process.

3. **Helps in high-impact factors:** Helps in identifying the factors that have the greatest impact on the cost and effort of a software project.

4. **Helps to evaluate the feasibility of a project:** This can be used to evaluate the feasibility of a software project by estimating the cost and effort required to complete it.

Disadvantages of the COCOMO Model

1. **Assumes project size as the main factor:** Assumes that the size of the software is the main factor that determines the cost and effort of a software project, which may not always be the case.

2. **Does not count development team-specific characteristics:** Does not take into account the specific characteristics of the development team, which can have a significant impact on the cost and effort of a software project.

3. **Not enough precise cost and effort estimate:** This does not provide a precise estimate of the cost and effort of a software project, as it is based on assumptions and averages.

Best Practices for Using COCOMO

1. **Recognize the Assumptions Underpinning the Model:** Become acquainted with the COCOMO model's underlying assumptions, which include its emphasis on team experience, size, and complexity. Understand that although COCOMO offers useful approximations, project results cannot be predicted with accuracy.

2. **Customize the Model**: Adapt COCOMO's inputs and parameters to your project's unique requirements, including organizational capacity, development processes, and industry standards. By doing this, you can be confident that the estimations produced by COCOMO are more precise and appropriate for your situation.

3. **Utilize Historical Data:** To verify COCOMO inputs and improve estimating parameters, collect and examine historical data from previous projects. Because real-world data takes project-specific aspects and lessons learned into account, COCOMO projections become more accurate and reliable.

4. **Verify and validate:** Compare COCOMO estimates with actual project results, and make necessary adjustments to estimation procedures in light of feedback and lessons discovered. Review completed projects to find errors and enhance future project estimation accuracy.

5. **Combine with Other Techniques:** To reduce biases or inaccuracies in any one method and to triangulate results, add COCOMO estimates to other estimation techniques including expert judgment, similar estimation, and bottom-up estimation.

Conclusion

For both software engineers and project managers, COCOMO is an applicable and useful tool at a time when effective project planning is essential to success. Its continued application and adaption in a variety of settings show how valuable it is in the always-changing field of software development.

Last Updated : 14 Mar, 2024

SWE 402 Lecture Note Draft 5d More materials on COCOMO model

Brief history of the COCOMO model

The COCOMO (Constructive Cost Model) model has a rich history rooted in the evolution of software engineering practices and the need for accurate software cost estimation. Here's a brief overview of its development and advancements:

### 1. **Origins and Development of COCOMO (1981)**

- **Barry W. Boehm**: The original COCOMO model was developed by Barry W. Boehm and published in his 1981 book, "Software Engineering Economics."

- **Purpose**: It was designed to provide a systematic way to estimate the cost, effort, and schedule for software development projects based on project size measured in KLOC (thousands of lines of code).

- **Models**: The original COCOMO model introduced three levels of estimation:

  - **Basic COCOMO**: A simple model that estimates effort and cost based solely on the size of the software.

  - **Intermediate COCOMO**: Includes additional cost drivers to account for various attributes of the software project.

  - **Detailed COCOMO**: Further refines the estimation by considering the impacts of different phases of the software development lifecycle.

### 2. **Key Features of Original COCOMO**

- **Project Types**: Categorizes projects into three types—organic, semi-detached, and embedded—based on complexity, team experience, and project constraints.

- **Effort Estimation**: Uses a formula involving constants and the size of the software to estimate effort in person-months.

- **Cost Drivers**: Considers 15 cost drivers in the intermediate model to adjust estimates based on factors like product complexity, team capability, and required reliability.

### 3. **Limitations and Need for Evolution**

- **Technological Advances**: The original model was designed for the software development practices and technologies of the early 1980s, which became outdated with the advent of new development methodologies and tools.

- **Software Reuse and Components**: The increasing use of reusable software components and off-the-shelf solutions required more sophisticated estimation techniques.

- **Agile and Iterative Processes**: The rise of agile and iterative development methodologies highlighted the need for a more flexible and adaptive estimation model.

### 4. **COCOMO II (1990s)**

- **Development**: In the 1990s, Barry Boehm and his team at the University of Southern California began developing COCOMO II to address the limitations of the original model.

- **New Features**: COCOMO II includes enhancements to handle modern software development practices, including:

  - **Early Design Model**: Provides rough estimates early in the project lifecycle when detailed information is not yet available.

  - **Post-Architecture Model**: Offers detailed estimates based on a comprehensive understanding of the system's architecture and components.

  - **Scale Factors and Cost Drivers**: Introduces 5 scale factors and 17 cost drivers to improve accuracy and reflect current practices.

  - **Productivity Improvements**: Accounts for productivity improvements from modern tools, techniques, and processes.

### 5. **Modern Adaptations and Use**

- **Continuous Updates**: COCOMO II continues to be refined and updated to reflect changes in technology and practices.

- **Widespread Adoption**: It is widely used in academia and industry for project estimation, budgeting, and planning.

- **Integration with Tools**: Many project management and estimation tools integrate COCOMO II to provide automated and accurate estimation capabilities.

### Summary

The COCOMO model has evolved significantly from its origins in the early 1980s to address the changing landscape of software development. Its evolution from the original COCOMO to COCOMO II reflects ongoing efforts to provide accurate, flexible, and reliable cost estimation methods that can adapt to modern software engineering practices and technological advancements.

DIFFERENCE AMONG THE 3 PRIMARY VERSIONS OF THE COCOMO MODELS

(That is the Basic, Intermediate, and Detailed COCOMO models)

The COCOMO (Constructive Cost Model) has three primary versions: Basic, Intermediate, and Detailed. Each version increases in complexity and accuracy, incorporating more factors to improve the estimation of effort, cost, and schedule for software development projects. Here's an explanation of the differences between these models:

### 1. The Basic COCOMO

Purpose: Provides a rough estimate of project effort and cost.

Complexity: Simple and straightforward.

Factors Considered: Only the size of the software (measured in KLOC - thousands of lines of code).

Formula:

\[E = a \times (KLOC)^b \]

\[D = c \times (E)^d \]

- **E**: Effort in person-months

- **D**: Development time in months

- **a, b, c, d**: Constants that vary depending on the type of software project (organic, semi-detached, embedded)

- **KLOC**: Thousands of lines of code

Characteristics:

- **Organic Projects**: Small, simple projects with a small team.

- **Semi-detached Projects**: Intermediate projects with mixed team experience.

- **Embedded Projects**: Complex projects with stringent requirements.

### **2. The Intermediate COCOMO**

Purpose: Provides more accurate estimates by incorporating additional factors that influence effort.

Complexity: More detailed than Basic COCOMO.

Factors Considered: Size of the software (KLOC) and a set of 15 cost drivers.

Formula:

\[E = a \times (KLOC)^b \times \prod_{i=1}^{15} EM_i \]

- **E**: Effort in person-months
- **a, b**: Constants based on project type
- **EM_i**: Effort multipliers (cost drivers) that account for product, hardware, personnel, and project attributes

Cost Drivers Include:

- Product attributes: Required software reliability, database size, product complexity
- Hardware attributes: Execution time constraints, storage constraints
- Personnel attributes: Analyst capability, programmer capability, application experience, virtual machine experience, programming language experience
- Project attributes: Use of software tools, required development schedule

### 3. The Detailed COCOMO (also known as COCOMO II)

Purpose: Provides the most accurate and detailed estimates by considering the effects of individual project phases and more detailed cost drivers.

Complexity: Most detailed and complex of the three models.

Factors Considered: Size of the software, 17 cost drivers, and 5 scale factors. It also breaks down the project into individual phases.

Formula:

\[E = a \times (KLOC)^{b + 0.01 \times \sum SF_i} \times \prod_{j=1}^{17} EM_j \]

- **E**: Effort in person-months
- **a, b**: Constants based on project type

- **SF_i**: Scale factors that account for the scale of the project (e.g., precedentedness, development flexibility, architecture/risk resolution)

- **EM_j**: Effort multipliers (cost drivers) similar to Intermediate COCOMO but more refined

Additional Features:

- **Phase Sensitivity**: Effort is estimated for each phase of the software lifecycle (e.g., requirements analysis, design, coding, testing).

- **Reuse and Maintenance**: Factors in the cost of reusing existing software and maintaining the developed software.

- **Modern Practices**: Updated to include modern software development practices and different types of software projects, including those with significant reuse or maintenance activities.

### Summary

- **Basic COCOMO**: Simple, uses only KLOC, suitable for rough estimates.

- **Intermediate COCOMO**: More accurate, includes 15 cost drivers to account for various project attributes.

- **Detailed COCOMO (COCOMO II)**: Most detailed, includes 17 cost drivers and 5 scale factors, and provides phase-wise effort estimation, suitable for modern and complex projects.

These models allow project managers to choose the level of detail and accuracy they need based on the project's size, complexity**, and available information.**

USES OF THE COCOMO MODEL IN PROJECT MANAGEMENT AND PLANNING

The COCOMO (Constructive Cost Model) model aids in project management and planning by providing systematic and quantitative estimates for various aspects of software development projects. Here are several ways in which the COCOMO model contributes to project management and planning:

### 1. **Effort Estimation**

The COCOMO model helps estimate the total effort required to complete a software project in terms of person-months. By inputting parameters such as the size of the software (measured in KLOC, or thousands of lines of code), project managers can predict the amount of human resources needed.

### 2. **Cost Estimation**

Based on the effort estimation, the COCOMO model allows project managers to estimate the overall cost of the project. This includes the cost of labor, tools, and other resources. Accurate cost estimation is crucial for budgeting and financial planning.

### 3. **Schedule Estimation**

COCOMO provides formulas to estimate the development schedule (in months). Knowing the effort and the productivity of the team, managers can derive a realistic timeline for the project, helping in scheduling tasks and setting milestones.

### 4. **Resource Allocation**

With a clear understanding of the effort required, project managers can allocate resources more effectively. This includes assigning the right number of developers, testers, and other personnel at various stages of the project.

### 5. **Risk Management**

By identifying the factors that influence effort and cost, such as product complexity, required reliability, and team experience, COCOMO helps in assessing potential risks. Understanding these factors allows managers to take proactive measures to mitigate risks.

### 6. **Project Planning**

COCOMO assists in detailed project planning by breaking down the project into smaller, manageable components. This includes planning for various phases such as requirements analysis, design, coding, testing, and maintenance.

### 7. **Performance Measurement**

Project managers can use COCOMO estimates as a benchmark to measure actual project performance. By comparing estimated values with actual values, managers can identify deviations and take corrective actions.

### 8. **Communication and Justification**

The model provides a standardized way to justify the required effort and cost to stakeholders, including clients and upper management. This transparency helps in securing necessary approvals and resources.

### 9. **Improving Estimation Accuracy**

COCOMO II, an updated version of the original model, incorporates more factors and refined cost drivers to improve estimation accuracy, making it adaptable to modern software development practices and various project types.

### Example of COCOMO in Project Management

Suppose a project manager needs to estimate the effort for a new software project expected to be 100 KLOC in size. Using the Basic COCOMO model for an organic type project, with coefficients \(a = 2.4\) and \(b = 1.05\), the effort \(E\) can be calculated as:

\[E = 2.4 \times (100)^{1.05} = 2.4 \times 114.81 \approx 275.54 \text{ person-months} \]

Using this effort estimation, the manager can plan the number of developers needed, the duration of the project, and the budget required. Adjustments can be made based on the project's complexity, required reliability, and other cost drivers in the Intermediate or Detailed COCOMO models.

In summary, the COCOMO model provides a structured approach to estimating and managing software project resources, timelines, and risks, thereby enhancing the overall planning and execution of software development projects.