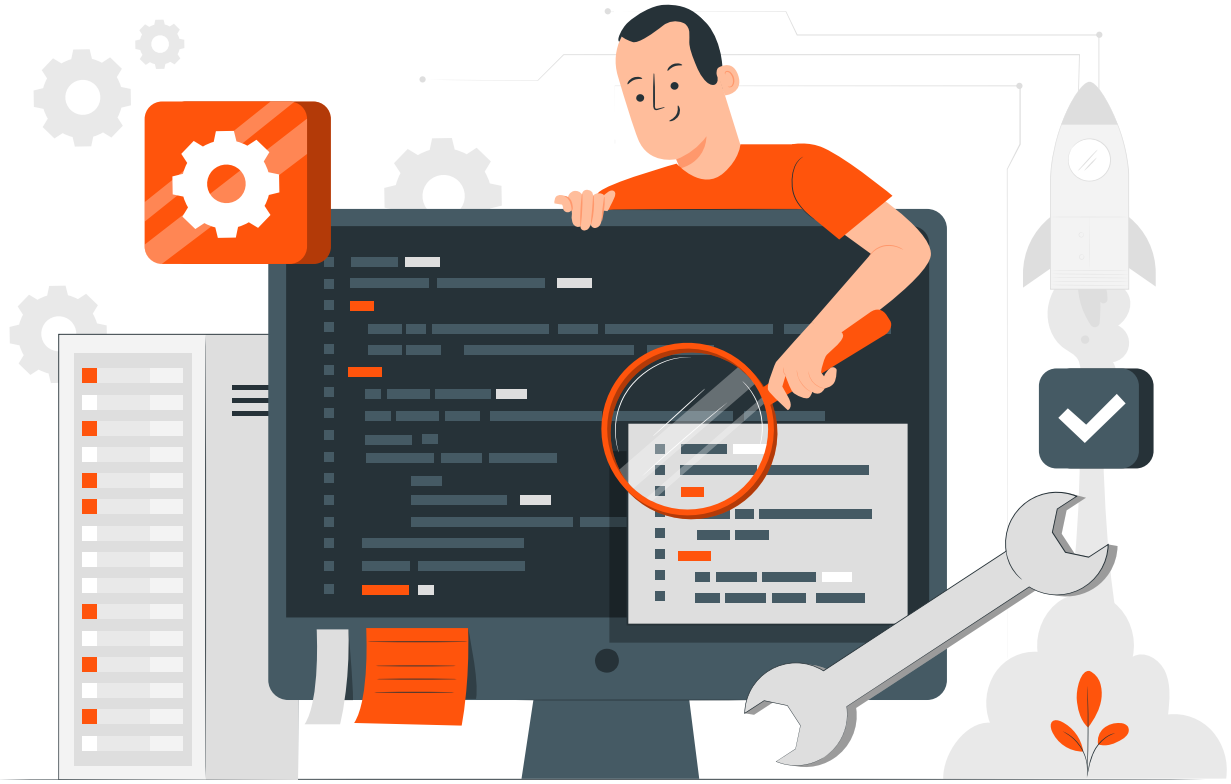
testbytes

Making Quality a Habit



History of Software Testing Estimation Models

INDEX

Introduction

The technique of estimating the work (expressed in person-hours or dollars) necessary to build or maintain software based on sparse, ambiguous, and noisy information is known as effort estimate in software testing. Project plans, iteration plans, budgets, investment evaluations, pricing procedures, and bidding rounds can all employ effort estimates as input. Knowing the workload and financial repercussions of performing a project is the major goal of software testing estimation encompassing effort and cost. Software testing cost estimation is a difficult task since there are so many variables that might influence the project's success. Data from previous projects with a comparable scope that were carried out and the individuals engaged in their implementation serve as the key component for estimating. The implementation of several strategies has been covered in this work, and an effort has been made to provide fresh concepts in order to arrive at accurate assessment of characteristics like cost and effort.

Key Words: - Software Cost Estimation, Artificial Intelligence, COCOMO, Wideband Delphi

Software Testing Estimation Models

Software Cost/Effort Estimation is a critical activity in software development projects that enables developers and managers to forecast, predict, and accurately quote the budget, schedule, and manpower in order to avoid overruns or underruns and to try to optimize the key factors that contribute to a project's success. Since 1950s, estimations of the time and money needed to build software have changed and are still being studied today.

For a specific software testing project in a specific environment utilizing certain methodologies, tools, and techniques, test estimation is the estimation of the testing size, testing effort, testing cost, and testing timeline. The term "testing size" refers to the total number of tests that must be run. Sometimes, especially in embedded testing (i.e., testing that is integrated into the software development process itself) and in circumstances when it is not required, this may not be calculated. The amount of effort required for testing is measured in either person days or person hours. Testing costs are the expenses necessary for testing, including the expense of human effort. While testing schedule is the duration in calendar days or months that is necessary for conducting the tests.

The following methods are available now for performing test effort estimation:

1 **Delphi / Wideband Delphi Technique (1950-60)**

2 **Analogy-Based Estimation (1970)**

3 **Software Size Based/Function Point Estimation (1979)**

4 **Constructive Cost Model (COCOMO) (1981)**

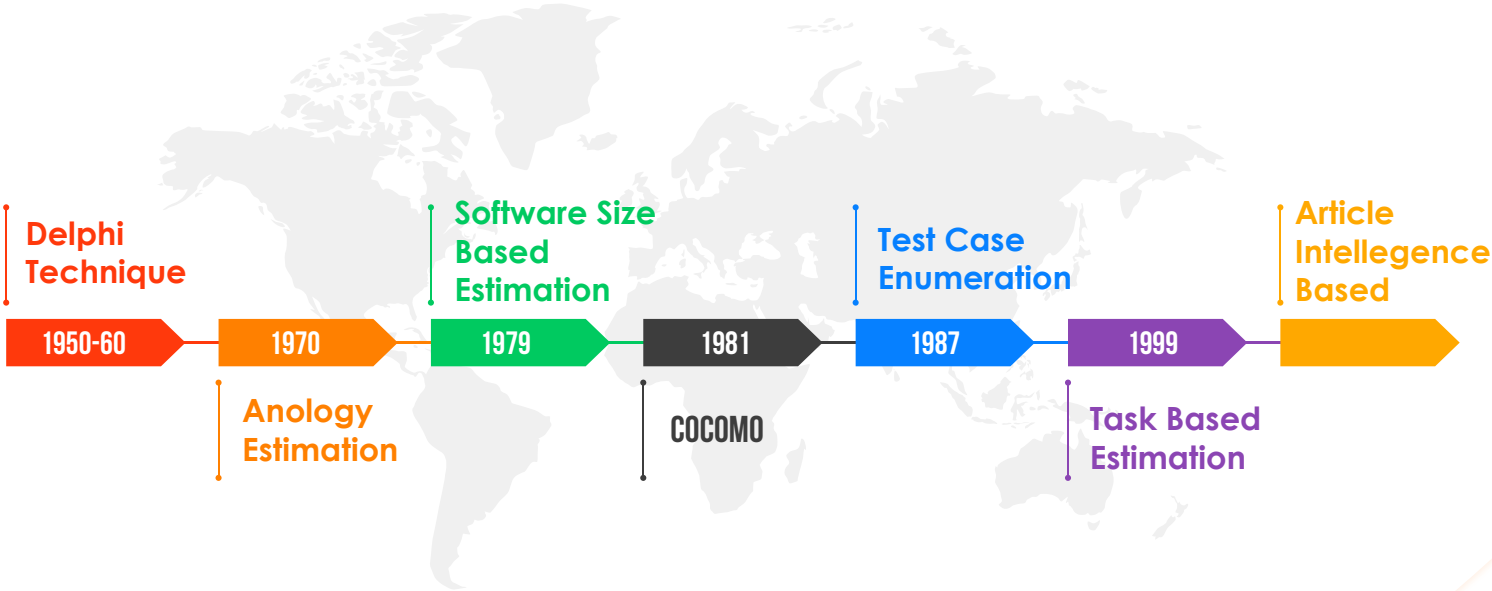
5 **Test Case Enumeration Based Estimation (1987)**

6 **Task (Activity) Based Estimation (1999)**

7 **Artificial Intelligence Based Estimation**

in calendar days or months that is necessary for conducting the tests.

Methods Available For Test Effort Estimation



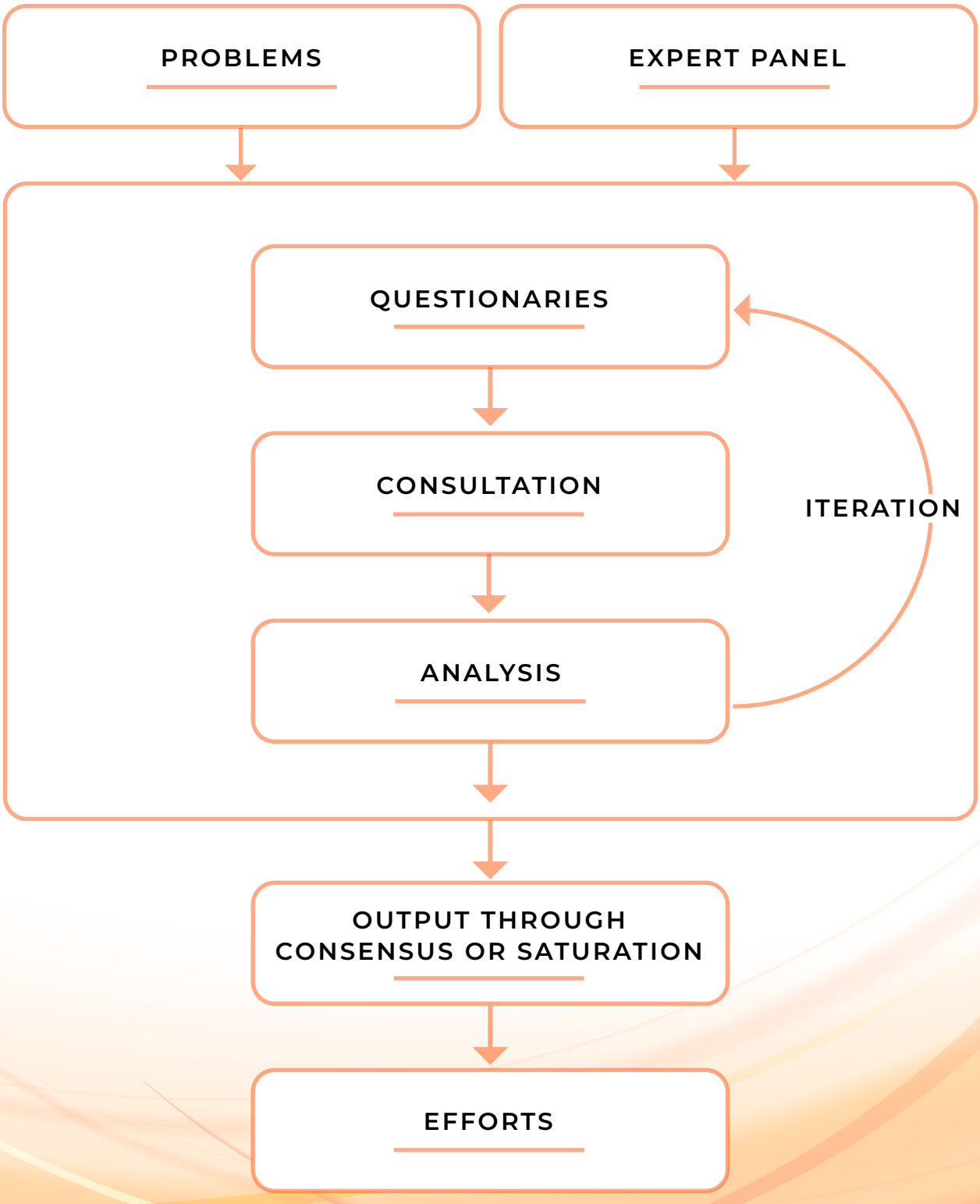
1 Delphi / Wideband Delphi Technique

The Delphi Method was created at the RAND Corporation in the 1950s-1960s. The following is one way the Delphi approach may be applied to software cost computation: Each estimator receives a copy of the System Definition paper and a form to record a cost estimate from a coordinator.

After reviewing the definition, the estimators do their anonymous calculations. They can dispute the coordinator, but they don't talk about their estimations between themselves.
A summary of the estimators' replies, including any odd justifications mentioned by the estimators, is prepared and distributed by the coordinator. Using the findings from the last estimate, estimators anonymously complete a new estimate. Those whose estimations drastically deviate from the group may be asked to justify their estimates.

The procedure was repeated as many times as necessary. Throughout the whole procedure, group discussions are not permitted. The method that is going to be discussed now is a modification of the classic Delphi procedure that improves communication while maintaining anonymity.

The coordinator creates a list of the estimations but leaves out any justifications. In order to focus on topics where the estimates are greatly disparate, the coordinator schedules a group meeting. Once more, estimators complete an estimate. Anonymously, the procedure is repeated as many times as required.


PROBLEMS
EXPERT PANEL
QUESTIONARIES
CONSULTATION
ANALYSIS
ITERATION
OUTPUT THROUGH CONSENSUS OR SATURATION
EFFORTS

Pros of the Delphi approach:

1. Can provide incredibly precise outcomes
2. Uses quite easily
3. Is regarded as a trustworthy procedure despite not being an ISO standard
4. Can be used for a variety of sectors or issues

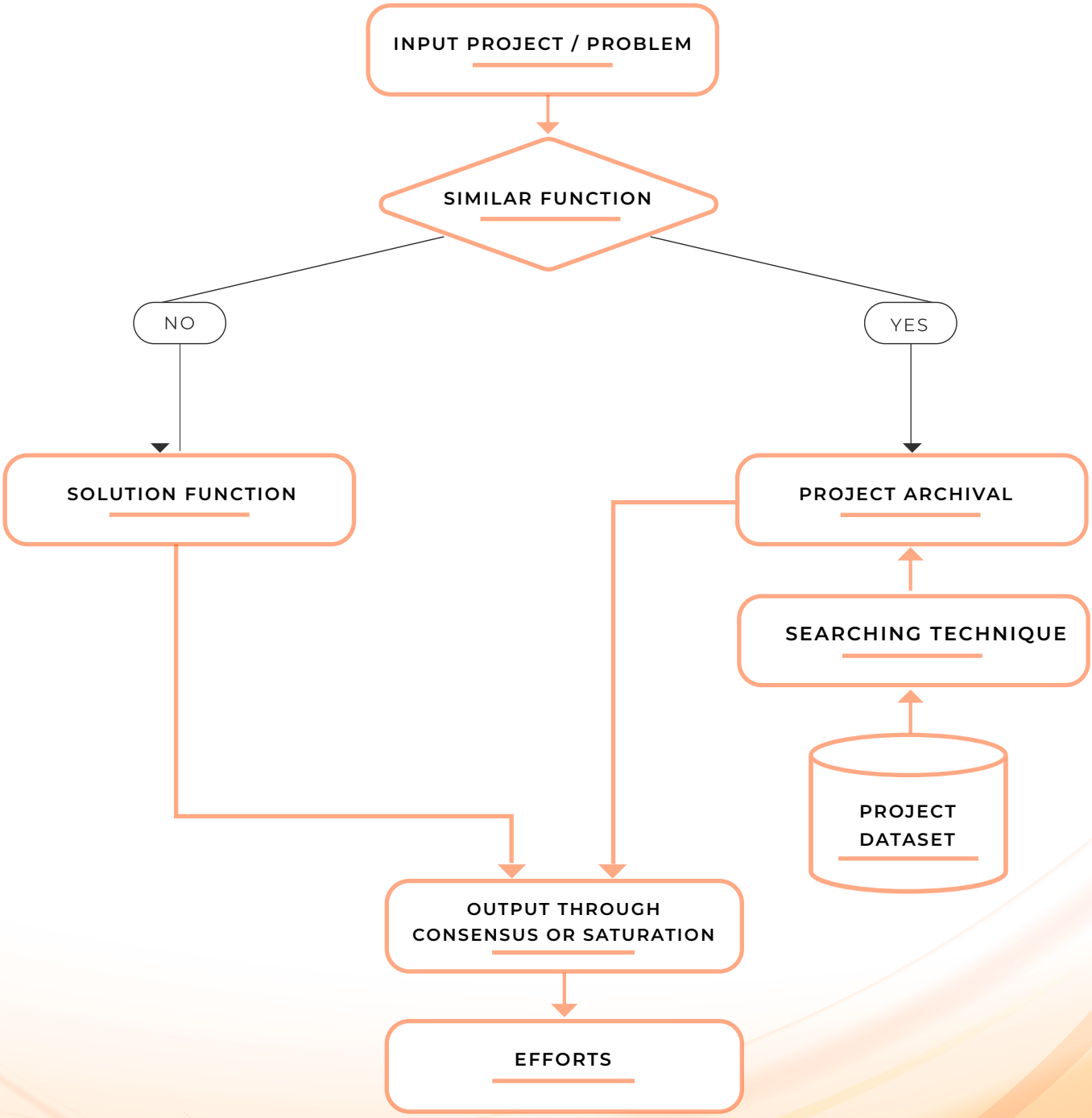
Cons of the Delphi method :

1. Need several subject-matter specialists
2. Might get pricey
3. Can take a lot of time
4. If the estimation's scope is excessively broad or poorly specified, it could not yield excellent results.

2 Analogy-Based/Top-Down Estimation

Analogy Based/Top-down design was proposed by IBM researchers Harlan Mills and Niklaus Wirth in the 1970s. Mills created and tested structured programming techniques in a 1969 attempt to automate the New York Times morgue index. The success of this project's engineering and management contributed to the adoption of the analogy-based/top-down method throughout IBM and the rest of the computer industry.
Analogous estimation is a technique that takes parameter values from previous data to estimate comparable parameters for a future activity. Examples of parameters include scope, cost, and duration. Scale examples include size, weight, and complexity. It is considered a blend of historical knowledge and expert opinion since the project manager's and maybe the team's expertise and judgment are used to the estimating process.
Analogous estimation uses similar prior project data to predict the length or expense of your present project, hence the term "analogy." When you have limited knowledge on your present project, you can utilize comparable estimate. Quite often, project managers will be required to provide cost and duration estimates for a new project because executives want decision-making data to determine whether the initiative is worthwhile. Typically, neither the project manager nor anyone else in the business has ever worked on a project like the new one, yet the executives still want precise cost and time predictions.
Analogous estimate is the best option in such instances. It is not perfect,

but it is accurate since it is based on historical facts. Analogous estimate is a simple approach to master. When compared to the early predictions, the project success rate might reach 60%.


INPUT PROJECT / PROBLEM
SIMILAR FUNCTION
NO
YES
SOLUTION FUNCTION
PROJECT ARCHIVAL
SEARCHING TECHNIQUE
PROJECT DATASET
OUTPUT THROUGH CONSENSUS OR SATURATION
EFFORTS

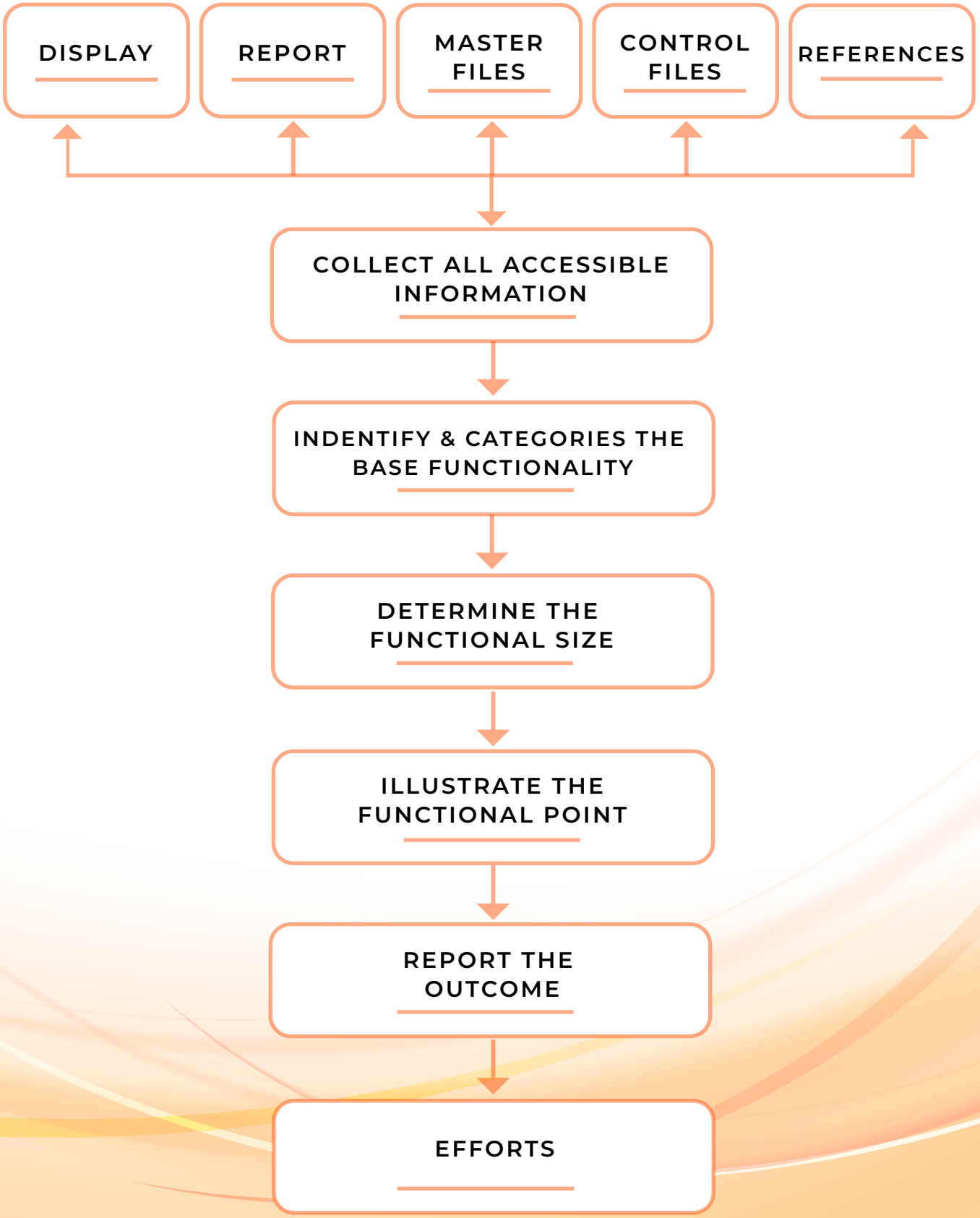
The Benefits of Analogous Estimation

1. Analogous estimate is a superior method of estimating in the early phases of a project when there are few specifics known.
2. The approach is straightforward, and the time required for estimation is minimal.
3. Because the approach is based on the organization's previous project data, the success rate may be predicted to be high.

Analogous estimating may also be used to estimate the time and effort required for particular jobs. As a result, while estimating tasks in Work Breakdown Structure (WBS), you may use Analogy.

3 Software Size Based/Function Point Estimation (1979)

This model was developed by Allan Albrecht at IBM in 1979 for Measuring Application Development Productivity. Software size would be accessible by the time a testing project is in its beginning stages. Now, we use this software size as the scope of the testing project. Then, in order to determine the needed work to carry out the testing project, we assign a Productivity number (rate of accomplishment) for the program size. By assuming that testing software with a size of one Function Point requires two person hours, we can determine the amount of work needed for the testing project based on the size of the program that has to be tested. If the program to be tested has a size of 1000 Function Points, then testing the software will require 2000 Person Hours based on the standard of 2 Person Hours per Function Point. However, as no standards body has any criteria for translating software size to effort, the company must establish & maintain those using historical data while strictly following a procedure. This standard must be generated & upheld for all software size measurements used by the organization.


DISPLAY
REPORT
MASTER FILES
CONTROL FILES
REFERENCES
COLLECT ALL ACCESSIBLE INFORMATION
INDENTIFY & CATEGORIES THE BASE FUNCTIONALITY
DETERMINE THE FUNCTIONAL SIZE
ILLUSTRATE THE FUNCTIONAL POINT
REPORT THE OUTCOME
EFFORTS

Benefits of Estimating Software Size

1. Extremely easy to learn and utilize
2. Quick to calculate the effort estimate - takes very little time.
3. The findings of effort estimation using this method may be remarkably accurate if the organization determines and maintains these standards using the correct process.

Demerits of Software Size-Based Estimation

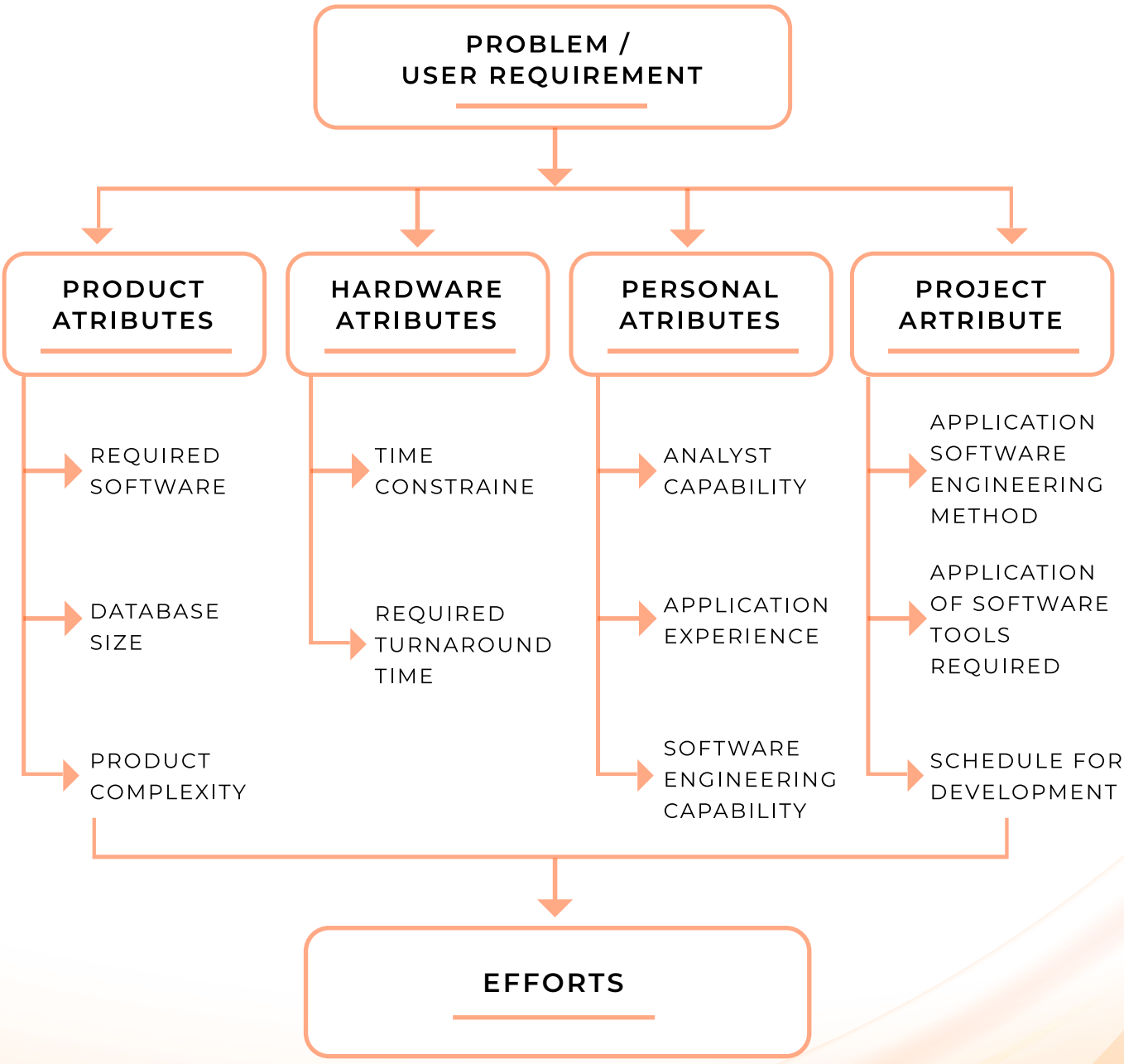
1. Too straightforward and un-auditable
2. Productivity cannot be calculated without testing size. On the other hand, software size may be used to calculate testing productivity.
3. Despite the fact that the size may be the same, testing requirements vary by application type. For instance, even if the size of a standalone software program and a web-based software application may be the same, they require different amounts of testing. Thus, the average program size may not always be appropriate.
4. Organizations must maintain precise records and hire full-time professionals to create and uphold standards. To effectively calculate these standards, the timesheets must be customized to collect the right information. Data collection must be meticulous.

4 Constructive Cost Model (COCOMO)

Barry W. Boehm created the Constructive Cost Methodology (COCOMO), a model for procedural software cost assessment. Line of Codes (LOC) was the primary measure used in calculating software development efforts or costs when Constructive Cost Model (COCOMO) was originally introduced in 1981 by Barry Boehm in his book Software Engineering Economics. This model is frequently referred to as COCOMO 81.

COCOMO II was created in 1995 and eventually published in the book Software Cost Estimation using COCOMO II in 2000. A bigger database was used to fine-tune COCOMO II, the successor to COCOMO 81, which is said to be more suitable for estimating contemporary software development projects. COCOMO II also supports more recent software development procedures. As software development technology changed from mainframe and overnight batch processing to desktop development, code reusability, and the usage of off-the-shelf software components, a new paradigm was required.
Three shapes that are more and more precise and intricate make up COCOMO. Because it lacks components to take into account differences in project parameters, Basic COCOMO's accuracy is only good for short, early,rough order-of-magnitude estimations of software costs (Cost Drivers). These Cost Drivers are taken into consideration by Intermediate COCOMO, while Detailed COCOMO also considers the impact of various

project phases. The last one is the Complete COCOMO model, which corrects both basic and intermediate inadequacies.estimations of soft-ware costs (Cost Drivers). These Cost Drivers are taken into consideration by Intermediate COCOMO, while Detailed COCOMO also considers the impact of various project phases. The last one is the Complete COCOMO model, which corrects both basic and intermediate inadequacies.


PROBLEM / USER REQUIREMENT
PRODUCT ATRIBUTES
HARDWARE ATRIBUTES
PERSONAL ATRIBUTES
PROJECT ARTRIBUTE
REQUIRED SOFTWARE
DATABASE SIZE
PRODUCT COMPLEXITY
TIME CONSTRAINE
REQUIRED TURNAROUND TIME
ANALYST CAPABILITY
APPLICATION EXPERIENCE
SOFTWARE ENGINEERING CAPABILITY
APPLICATION SOFTWARE ENGINEERING METHOD
APPLICATION OF SOFTWARE TOOLS REQUIRED
SCHEDULE FOR DEVELOPMENT
EFFORTS

5 Summaries

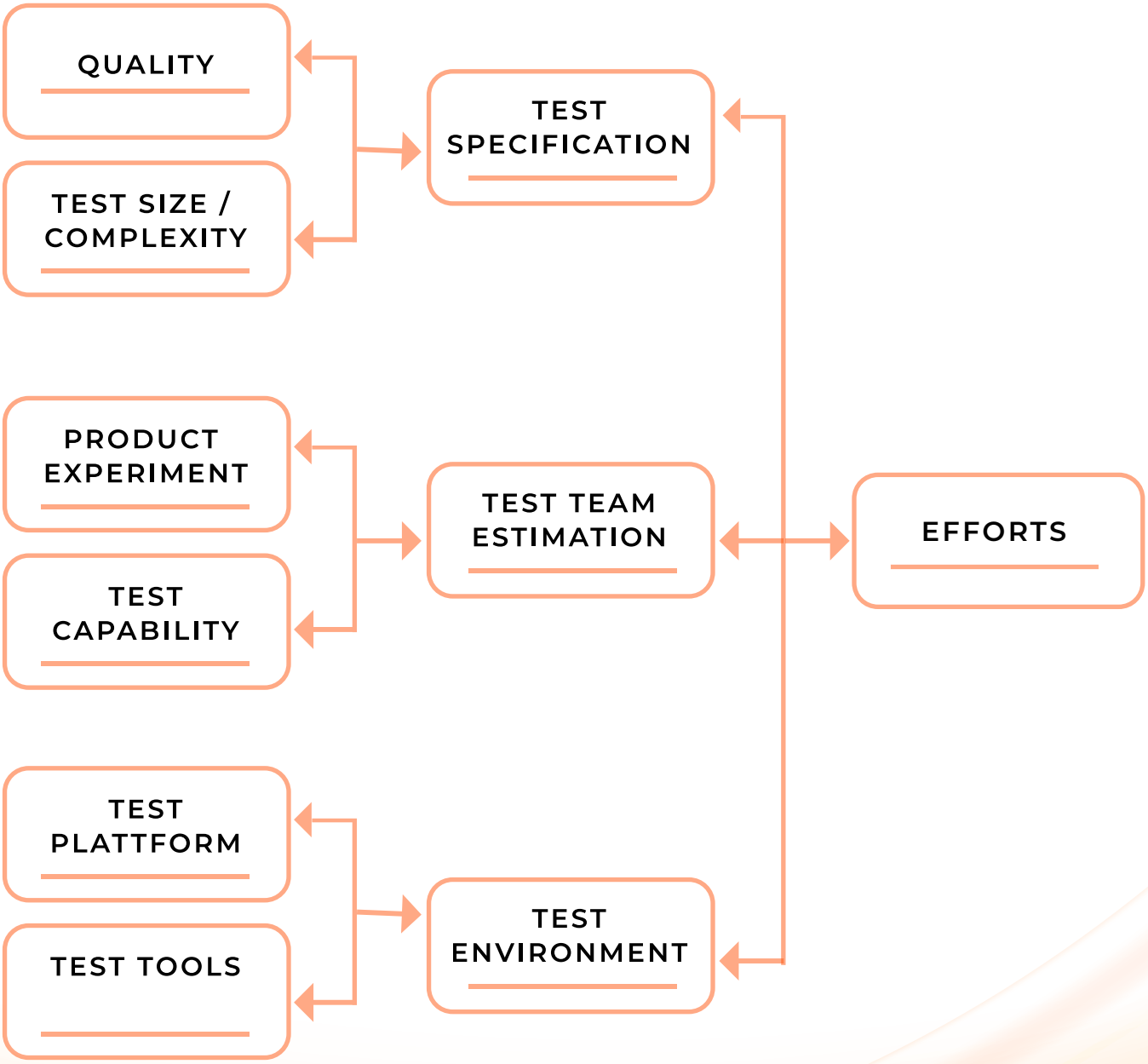
1. When to expect to get Project's anticipated effort estimate
2. Best-Case timeframes for estimating the best-case effort
3. Worst-Case scenarios for estimating the worst-case effort
4. Normal-Case timeframes to gain an estimate of the normal-case effort

- **Favorable Features of Test Case Enumeration Based Estimation The advantages of this method are as follows:**

1. Estimates that are auditable include enough information for a peer to evaluate them and make sure they are complete and as accurate as possible.
2. Fairly accurate - correctness is guaranteed because all test cases are listed and the estimated effort is calculated three times.
3. By marking the test cases as complete, progress tracking is made easier and the percentage of completion may be calculated.
4. Makes it easier to provide a range of values for the estimations, such as a. The project can be completed with an expected effort of so many person hours and a minimum effort of so many person hours. The decision-makers can now specify the negotiating margins in their quotations.

Test Case Enumeration Based Estimation's Demerits

1. There is no testing size, thus productivity cannot be calculated.
2. It takes time to finish the estimation since every test case and its associated overheads must be included.


QUALITY
TEST SIZE / COMPLEXITY
TEST SPECIFICATION
PRODUCT EXPERIMENT
TEST CAPABILITY
TEST TEAM ESTIMATION
EFFORTS
TEST PLATTFORM
TEST TOOLS
TEST ENVIRONMENT

6 Task (Activity) Based Estimation

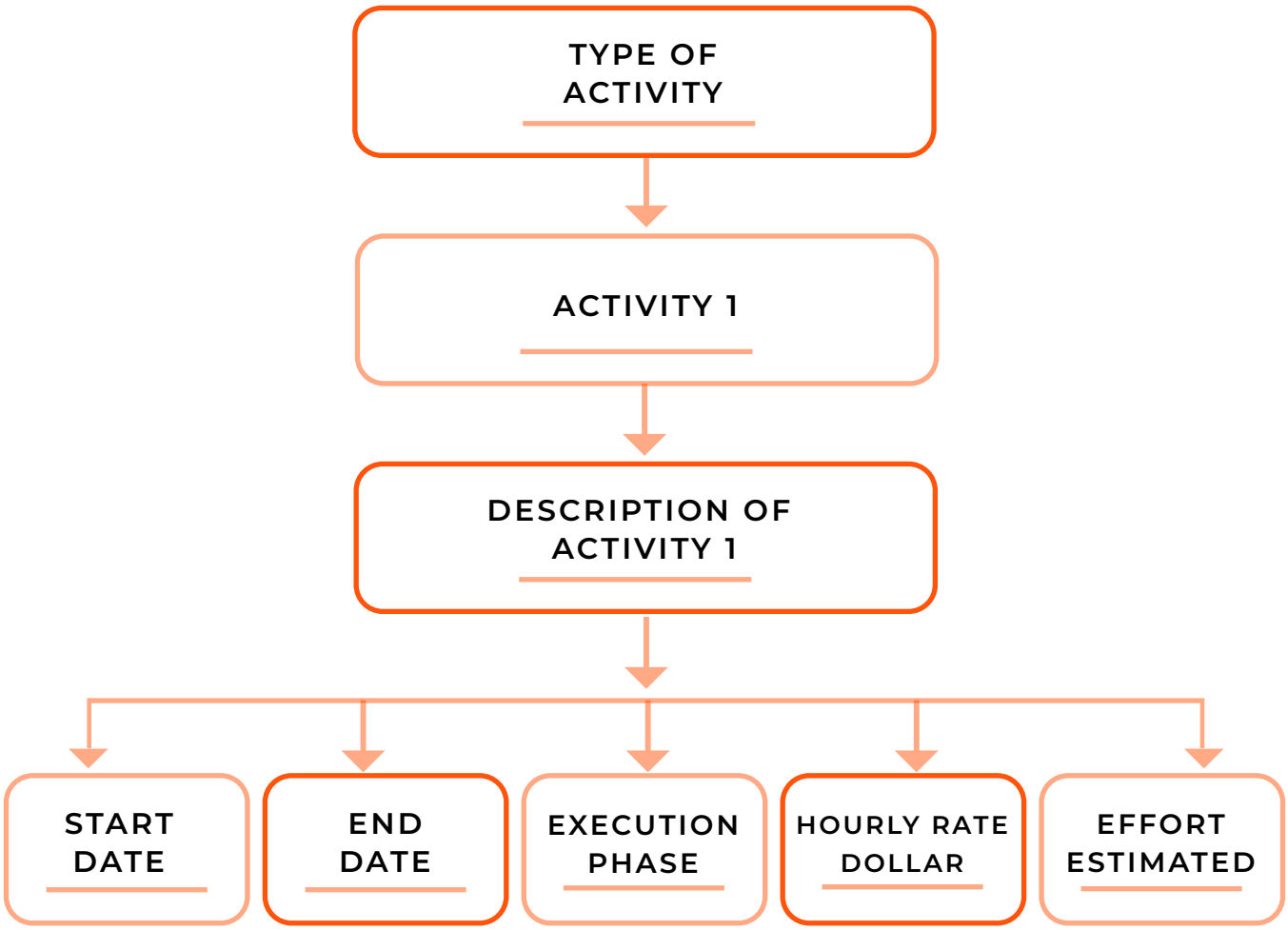
Proponents of the Balanced Scorecard, Robin Cooper and Robert S. Kaplan, attracted attention to these principles in a series of essays published in Harvard Business Review beginning in 1988. Cooper and Kaplan defined ABC as a method for addressing the shortcomings of existing cost management methods. Traditional costing methodologies are frequently unable to correctly assess the actual costs of manufacturing and related services. As a result, managers were making choices based on erroneous data, particularly when many goods were involved. Later, in 1999, Peter F. Drucker described activity-based costing in the book Management Challenges of the 21st Century. He claims that whereas activity-based costing also records the costs of not acting, such as the cost of waiting for a necessary item, traditional cost accounting just records the expenses of doing something, such as cutting a screw thread. Traditional cost accounting does not record expenses; activity-based costing does.

If any effort variation occurs, it will be able to measure at a specific activity level rather than having an impact on all of the activities if activity-based estimate is used to focus on essential activities. By gathering and examining the data for 12 Enhancements from Application service Maintenance projects that have already been provided, activity-based software estimating based on work break down structure has been described. This article describes how to achieve at precise estimation at vari

ous micro-level Software Development Life Cycle activities (SDLC).

The Work Break Down Structure (WBS) results in the division of a big component or activity into smaller activities or sub-components. The process of breaking things down will go on until it becomes impossible to do so physically or rationally for each lower level of subcomponents. It is necessary to assess and map each subcomponent and smallest action to a set of requirements. Since most application service maintenance projects entail making modest improvements, we can't always use the full promise of the estimation methodology, such as Function Point Analysis or lines of code (LOC). In this instance, the majority of the business uses work breakdown structure and activity-based software estimating.

WBS concentrates on segmenting projects into several activities and allocating resources to each sub activity. The division of tasks into several activities is not consistent
throughout applications or projects, and it differs fromone company to another according to the processes they have established.
By examining the predicted effort data of comparable projects that were conducted at the micro level of SDLC activities, it is necessary to anticipate different possible characteristics to increase the accuracy of WBS.


TYPE OF ACTIVITY
ACTIVITY 1
DESCRIPTION OF ACTIVITY 1
START DATE
END DATE
EXECUTION PHASE
HOURLY RATE DOLLAR
EFFORT ESTIMATED

The advantages of task-based effort estimation for testing projects

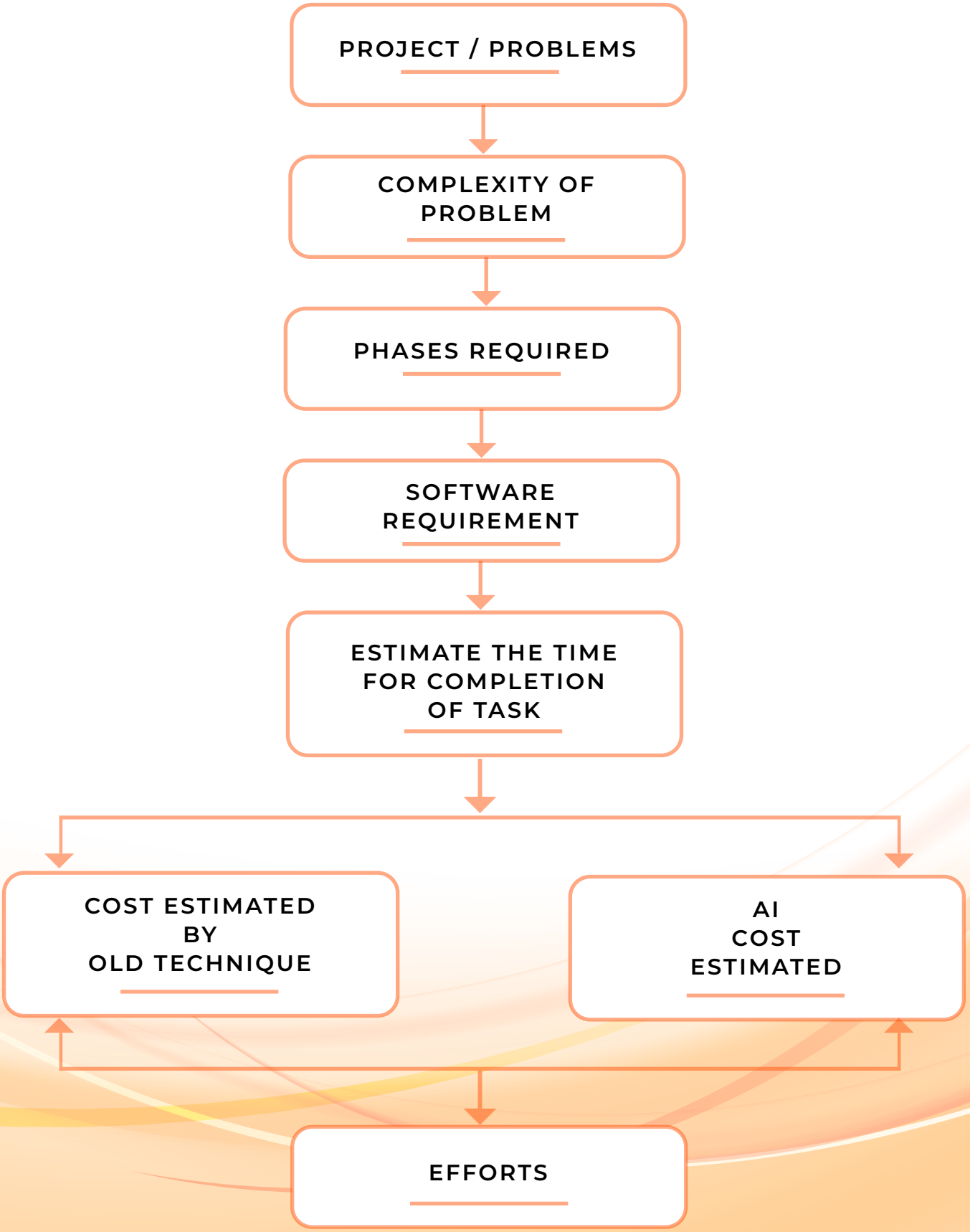
1. This best reflects how projects are carried out
2. This method provides the most accurate effort estimations by taking into account all the actions that are done
3. It includes enough information to allow for review, auditing, and post-mortem analysis when comparing it to the real values
4. It's quick and simple to create an estimate.
5. Makes it simple to track project progress by marking completed tasks and instantly calculating the percentage of completion
6. Applicable for use in estimating test effort using analogies

The Demerits of task-based effort estimation for testing projects

1. Because testing size is not calculated, testing productivity cannot be determined.

7 Artificial Intelligence based Estimation

In order to guarantee client satisfaction and repeat business, rapid and precise project cost estimating is essential. However, it continues to be one of the most difficult tasks in software engineering, particularly when dealing with complicated, large-scale, and innovative projects. The old technique of software cost assessment is being transformed into a flexible and intelligent approach thanks to breakthroughs in artificial intelligence technology. An intelligent system to locate and remove duplicates, assess and pinpoint ambiguity, and process in many formats with little to no human involvement is a compelling use case for quick and accurate project cost estimating.


PROJECT / PROBLEMS
COMPLEXITY OF PROBLEM
PHASES REQUIRED
SOFTWARE REQUIREMENT
ESTIMATE THE TIME FOR COMPLETION OF TASK
COST ESTIMATED BY OLD TECHNIQUE
AI COST ESTIMATED
EFFORTS

Challenges of Today's Project Cost Estimation

1. Failing to take unclear requirements into account
2. The tendency to overestimate production as a result of the assumption that all resources would perform at full capacity
3. Extensive time and effort wasted in finding and removing duplicate estimates
4. Adding padding to projections to account for unforeseen expenditures
5. Ignoring significant cost drivers under pressure from stakeholders to release the statistics quickly
6. Enduring on technical progress and shifting market dynamics

Benefits

1. Quicker project estimating by automation of tedious activities
2. Compared to conventional approaches, there is a 30–40% increase in precision
3. Accurate numbers supported by past evidence
4. Timely estimate of complex projects with numerous factors and enormous databases
5. Automatic cost-driver detection and evaluation
6. There should be some provision for handling ambiguous needs, such as missing values and incomplete databases
7. Creation of reusable, trained algorithms for estimations in the future
8. The capacity to provide estimates for documents with a range of scopes, traits, and formats

Conclusion

It is concluded that all the above-mentioned testing techniques have some limitations in different cases. Some of them have size related problems, some fail to calculate productivity, some are costly and time-consuming, etc. To address the issue of software test effort estimate, Testbytes developed a test effort calculator for cost estimates, which is used to determine the amount of time and effort necessary to test your product. The software test effort calculator from Testbytes is intended for specification and user preferences. The test cost calculator covers a wide range of topics, including banking and finance, telecommunications, e-commerce, and so on. The cost calculator is platform agnostic, which means you may use it on web, mobile, or both at the same time. The ultimate cost is determined by the total number of testing cycles necessary for the entire procedure.

References

1. Murali Chemuturi "Test Effort Estimation".
2. Rodríguez Montequín, V.; Villanueva Balsera, J.; Alba González, C.; Martínez Huerta, G." Software project cost estimation using AI techniques" Proceedings of the 5th wseas/iasme Int. Conf. on systems theory and scientific computation, Malta, September 15-17, 2005 (pp289-293)
3. Sheikh Umar Farooq, SMK Quadri "Empirical Evaluation of Software Testing Techniques – Need, Issues and Mitigation" Software Engineering : An International Journal (SEIJ), Vol. 3, No. 1, April 2013
4. Dheeraj Kapoor, R. K. Gupta "International Journal of Research and Development in Applied Science and Engineering (IJRDASE) ISSN: 2454-6844"