International Journal of Technical Research and Applications e-ISSN: 2320-8163,
www.ijtra.com, Volume 5, Issue 2 (March - April 2017), PP. 100-103

CASE BASED APPROACH FOR SOFTWARE COST ESTIMATION

[1] Gyanendra Singh, [2] Dr Anshu Srivastava, [3] Dr S.Q.Abbas

[1]Research Scholar , [2]Asst Professor ,[3] Professor and Director

[1,2] Shri Venkateshwara University, Gajraula, UP, India.

[3] AIMT Lucknow UP, India,

[1] gyanendrasingh@satnacement.com, [2] anshu_qrat@yahoo.co.in, [3] qrat_abbas@yahoo.com

***Abstract* — Reliable and accurate software cost estimation (SCE) is essential for effective project management. During the past two decades, many software cost estimation techniques are utilized to predict the cost. A common weakness of most models is their limited usefulness to predict the cost accurately at an early stage of the software development life cycle. In this paper we introduce a new cost estimation (CE) model; using the case-based reasoning tool, a model with "just enough" of each attribute to satisfy the desires of the system.**

***Index Terms* — Approach for software, cost estimation techniques.**

I. INTRODUCTION

It is the intention of this work to utilize the specified features representing the non-functional requirements i.e. quality attributes defined by the project managers, consequently the functional requirements since the former affects the latter as for example, a user's requirement not to be obstructed by a slow system performance in conducting a task should be translated into a requirement on transaction throughput and as another example the web based catalogs services which have on line payment options should be accompanied with a secure electronic transaction payment requirement that would acquire policies like cryptographic controls for the protection of the information transmission, as the basis for software cost estimation (SCE) in order to provide enhanced and more realistic results when undertaking the cost estimation process. Recent research has focused on the use of analogy is searching databases . Analogy has the benefits of 1) it is easily understood as a method, 2) it is easy to apply, and 3) it is applicable to small and large datasets; generic, industry specific or organization specific. Nowadays, there has been continued and increasing attention on the quality of software and the need to develop quality software products within cost and time limitations became of high priority because of the following reasons as adapted from:

1) A software product that is so unsatisfactory and unsuitable for the purpose intended that it has to be replaced with a new product; the company bears the cost of replacing the software product and the cost of the new product;

2) A software product that is so difficult to learn and use or that cannot be adapted to suit the users preferences and skills that it results in loss of time and consequently loss of productivity; the company bears the labor cost of the lost time;

3) A software product that has such a long learning curve that considerable time is lost in learning to use the product; the company bears the labor cost for the extra learning time;

4) A software product that fails frequently and requires fixing at each failure; the company bears the amount of rework to correct faults and the cumulative cost of doing so.

The main objectives of this work are to:

1. Identify a method for determining the software features based on the quality attributes that can be specified and quantified by the software project managers in the early phase of software development;

2. Introduce a new cost estimation (CE) model; using the case-based reasoning tool, a model with "just enough" of each attribute to satisfy the desires of the system.

The rationale for the usage of the case-based reasoning tool is to characterize the project for which the estimate is to be made, relative to a number of quality attributes. This description is then used to find other similar already finished projects, and an estimate for the new project is made based on the known effort values for those finished projects. Similarity of cases is determined by the specified set of project quality attributes which make the case different from others.

II. SOFTWARE QUALITY

"Quality comprises all characteristics and significant features of a product or an activity which relate to the satisfying of given requirements". "Quality is the totality of features and characteristics of a product or a service that bears on its ability to satisfy the given needs". "The totality of features and characteristics of a software product that bear on its ability to satisfy given needs: for example, conform to specifications; The degree to which software possesses a desired combination of attributes; The degree to which a customer or user perceives that software meets his or her composite expectations; The composite characteristics of software that determine the degree to which the software in use will meet the expectations of the customer".

Another model is the ISO/IEC 9126, which classifies software quality into four categories and these are:

- Process quality, which is the quality of software lifecycle processes;
- Internal quality, which is the quality of intermediate products, including static and dynamic models, documentation and source code;
- External quality, which is the quality of the final system as assessed by its external behavior;
- Quality in use, which is the effect of the system in use that is the extent to which users can achieve their goals using the system.

According to this approach, the ISO 9126 model splits software quality into six quality characteristics: functionality, reliability, usability, effectiveness, maintainability and portability. Each software quality characteristic is defined as a set of attributes that are supported by a relevant aspect of the software .

The advantages, or in other words, the most important features of the model are that:

- These approach to quality evaluation decomposes the concept of quality into a set of lower level quality characteristics that are recognizable properties of a product or service which refine ‘‘quality’’ into something more concrete and measurable;
- It defines the internal and external quality characteristics of a system, where these internal attributes of the software influence or determine the external attributes obtained by the end-user and that it is generic so it can be applied to any software product by tailoring to a specific purpose;
- It defines a three-level (strict) hierarchical structure of quality concepts;
- Familiar labels or single words are used to identify each characteristic and subcharacteristic, using terms that are commonly understood in practice;
- It consists of concise definitions, where each characteristic and subcharacteristic is defined using a single sentence ;
- Even evaluation procedures are defined in a separate standard to illustrate procedures for conducting product evaluations;
- Finally ISO/IEC 9126 is preferable because it represents a broad consensus among researchers and practitioners and is widely accepted and used in practice.

“Quality” in this sense comprises some set of key behavioral attributes, which are reliability (R), performance (P), fault tolerance (F), safety (Sa), security (Se), availability (A), testability (T), and maintainability (M). Software quality (Q) is thus a function of these combined attributes plus an error term (ε) i.e. quality would be Q = f(R, P, F, Sa, Se, A, T, M) + ε.

If quality can change quickly relative to new threats or changes in the operating environment, then quality is based not only on the attribute set but also on the execution environment. This in turn means that the environment can potentially modify any attribute’s value for a given piece of software. Thus, achieving quality requires weighting each attribute according to its importance to the system into which the software will be embedded. Each of the eight attributes is given a wi value in the range of zero to one and place them in this linear equation: Q = wRR +wPP + wFF + wSaSa + wSeSe + wAA + wTT + wMM, where the sum of the weights is1.0. It is assumed that it is a linear equation as it is believed that as long as any of those attributes increases (i.e. be highly attained in the software project), this consecutively improves quality as well. The weighting for each attribute depends on the type of software. For a financial system, the weight for security would probably be higher than that for safety. For a safety-critical system, the key attributes would probably be reliability, performance, safety, fault tolerance, and availability. For an e-commerce system, the key weighted attributes would be reliability, performance, availability, security, and maintainability (maintenance on these sites occurs continually).

III. Justifying The Selection Of The Following Quality Attributes

When software is seen to exhibit undesirable behavior, the data which is processed, the machinery on which the software runs, and by extension the people dealing with those machines might be negatively affected. Software errors have major consequences, sometimes may cause loss of life such as in aircraft flight control systems and nuclear systems. The quality requirements needed for a software product relies on the environment that it would implied into. The most important attributes that should be considered at the analysis of the project requirements are: usability, maintainability, performance, testability, reliability, and security. This means that a software product should possess these attributes but with relatively different percentages of importance according to the software classification type.

Usability: is defined as "The capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions". It is related to the "set of attributes that bear on the effort needed for use, and on the individual assessment of such use, by a stated or implied set of users". Usability is the "effort required to learn, operate, prepare input, and interpret output of program", "the ease with which members of a specified set of users are able to use something effectively.

Maintainability: is the extent to which updating the software is facilitated to satisfy new requirements without affecting the operability of the software system, i.e. changing functions, correcting functions, adding and deleting functions must be easy to accomplish. “The capability of the software product to be modified. Modifications may include corrections, improvements or adaptation of the software to change in environment, and in requirements and functional specifications", "effort required to locate and fix an error in an operational program".

Performance: is the key attribute for a software system applications, it expresses its functionality. It has a practical significance as it describes the number of operations that can be done or completed in a given period of time and the delay between the user operation request and its completion as to ensure the minimum number of customers and transactions that the software system is capable of serving and dealing with them effectively. It is "measured by evaluating processing speed, response time, resource consumption, throughput, and efficiency". Efficiency is "the degree to which something effectively uses (i.e. minimizes its consumption of) its resources. These resources may include all types of resources such as computing (hardware, software, and network), machinery, facilities, and personnel". Performance could not be adequately verified without testability.

Reliability: is significant to determine the extent to which the software system can be expected to perform its intended functions satisfactorily under stated conditions as the reliability parameter value greatly affects the development cost to both the customer and the producer. This entails a time factor in that a reliable product is expected to perform correctly over a period of time. It cannot be evaluated in its own right; there are related attributes to reliability, which are indeed needed to be measured such as the mean time between failures (MTBF) that can also be specified as the number of failures during a given period or the failure rate, reliability increases as the (MTBF) increases (i.e. the number of faults goes down), and also availability of the system which is the proportion of time a system is in a functioning condition that can be defined as the probability that the system is operating at a specified time, especially needed in applications such as web based.

Security: establishes how well a system resists penetrations from outside and misuse by insiders, it concerns transcend the boundaries of an automated system. It is defined as "the capability of the software product to protect information and data so that unauthorized persons or systems cannot read or modify them and authorized persons or systems are not denied access to them", "the availability of mechanisms that control of protect programs and data". A secure system is a system which does exactly what we want it to do and nothing to do even when someone else tries to make it behave differently. Security has to be compared and contrasted with other concepts such as: performance, reliability because an extremely secure system may affect the reliability of the system and its performance.

IV. Need for software cost estimation

Cost estimating, risk analysis, project scheduling, quality management planning and change management planning are the five major for software project planning. Our attempt to measure how much money, effort, resource and time will take to make a specific software based system or product. Estimating is important if you make a car without knowing much you were spend the tasks you required to do it and the time limit for the work to be completed. For that we required or develop an estimate before we start making the software. After survey from various papers and review report we got the following statistics:-

1) Near about 15% of the projects get cancelled before their completion.

2) Successful project rate is about 34% i.e. it gets completed on time and on budget.

3) 20-25% projects don't meet Return on Investment.

4) Directly or indirectly approx 20% get failed at the initial phase.

V. Case Based Reasoning Approach For Software Cost Estimation

Case based reasoning is a method of machine learning that seeks to emulate human recollection and adaptation of past experiences in order to find solutions to current problems. That is, as humans we tend to base our decisions not on complex reductive analysis, but on an instantaneous survey of past experiences ; i.e., we don't think, we remember. CBR is purely based on this direct adaptation of previous cases based on the similarity of those cases with the current situation. Having said that, a CBR based system has no dedicated world model logic, rather that model is expressed through the avail-able past cases in the case cache. This cache is continuously updated and appended with additional cases.

Aamodt & Plaza describe a 4-step general CBR cycle, which consists of:

1) Retrieve: Find the most similar cases to the target problem.

2) Reuse: Adapt our actions conducted for the past cases to solve the new problem.

3) Revise: Revise the proposed solution for the new problem and verify it against the case base.

4) Retain: Retain the parts of current experience in the case base for future problem solving.

VI. Accuracy Improvement For Cost Estimation

Cost estimation is a critical issue for software organizations. The need is to get the best estimate when planning a new project. Improving the prediction capability of software organizations is a way of improving their competitive advantage. Prediction is a major task when trying to better manage resources, mitigate the project risk, and deliver products on time, on budget and with the required features and functions. Estimates can help us make decisions that are more informed if, and only if, they can rely on the results to be accurate. If the results are accurate if it is reliable (correct) and valid (stable). Better estimates can be obtained by improving the estimation model. An estimation model is composed of some input variables (explanatory or independent variables), one output variable (the estimate or the dependent variable), and a function that calculates the outputs from the inputs. There are many ways of improving the estimates. For instance, we can choose:

1) a better function (e.g., the one that describes more appropriately the relationship between inputs and output), and/or

2) more explanatory input variables. In the former case, we can choose the type of function, e.g., linear or logarithmic that fits best.

In fact, a more parsimonious model (with fewer parameters) is preferable to one with more parameters because the former is able to provide better estimates with the same number of observations. This task can be performed in many ways, e.g., shrinking the input set into an equivalent pattern or removing irrelevant variables (e.g., stepwise methods). We can use Curvilinear Component Analysis (CCA) as a input shrinkage technique, which produces (shrunken) data sets where we apply ordinary least squares (OLS)

VII. Development Of A Case Based Realistic Model For Cost Estimation Of Any Software

Software cost estimation steps

Software cost estimation process involves basic seven steps:-

1) Demonstrating the specific goals.

2) Generating a sketch for required data and resources.

3) Gathering the software requirements.

4) Checking the feasibility of the software.

5) Using various cost estimation techniques to estimate the cost.

6) Balancing different estimates and restate the process.

7) Monitoring the estimated output and carry out the above steps.

This is a set of techniques and procedures that is used to derive the software cost estimate by using set of inputs to the process and then it will generate the outputs based on the given inputs.

Conclusion

Accurate software cost and schedule estimation are essential for software project success. Often it referred to as the "black art" because of its complexity and uncertainty, software estimation is not as difficult or puzzling as people think. In fact, generating accurate estimates is straightforward-once you understand the intensity of uncertainty and framework for the modeling process. The mystery to successful software estimation-distilling academic information and real-world experience into a practical guide for working software professionals. A proven set of procedures, understandable formulas, and heuristics that individuals and development teams can apply to their projects to help achieve estimation proficiency with choose appropriate development approaches First, the estimate of the size is converted into an estimate in nominal man-months of effort. As this nominal effort takes no advantage of knowledge concerning the specific characteristics of the software product, the way the software-product will be developed and the production means, a number of cost influencing factors (cost drivers) are added to the model. The effect of these cost drivers must be estimated. This effect is often called a productivity adjustment factor. Application of this correction factor to the nominal estimation of effort provides a more realistic estimate.

References

[1] Thibodeau, Robert. An Evaluation of Software Cost Estimating Models. Huntsville AL: General Research Corporation, 1981.

[2] IIT Research Institute. Test Case Study: Evaluating the Cost of Ada Software Development. Langham MD: IIT Research Institute, 1989.

[3] Bernheisel, Wayne A., "Calibration and Validation of the COCOMO II.1997.0 Cost/Schedule Estimating Model to the Space and Missile Systems Center Database." Unpublished masters thesis. Dayton, OH, Air Force Institute of Technology. 1997.

[4] Ram D. J. and S. V. G. K. Raju, “Object Oriented Design Function Points”, IEEE , The First Asia-Pacific Conference on Quality Software (APAQS'00), October 2000, pp. 121

[5] Sneed, H., “Estimating the development costs of object oriented software”, in: Proceeding of 7th European software control and metrics conference, 1996 ,Wilmslow, UK.

[6] Mehler, H. and A. Minkiewicz, “Estimating Size for object oriented

[7] G. Teologlou. Measuring object oriented software with predictive object points. In 10th Conference on European Software Control and Metrics, May 1999. Available at http://www.escom.co.uk/publications. software”, in: Proceeding of ASM’97 Application in Software measurement, Berlin, 1997.

[8] Caldiera, G. , G. Antoniol, R. Fiutem, C. Lokan,” Definition and Experimental Evaluation of Function Points for Object-Oriented Systems”,IEEE, 5th. International Symposium on Software Metrics , March 1998, pp. 167.