Test Effort Estimation Techniques

Snehil Shwetabh 2011C6PS502P
Harshit Gupta 2011C6PS837P
Akhil Tripathi 2011C6PS739P

Arpit Phanda 2011C6PS613P
Nitin Suri 2011C6PS802P
Nihav Jain 2011C6PS719P

***Abstract:*-Software Testing is a very vital part of Software Development life Cycle.**
It consist of estimating testing effort, selecting suitable test team, designing test cases, executing the software with those test cases and examining the results produced by those executions. Testing ensures good software quality. Development Cost and the success of a software is dependent on testing it. Hence the paper proposes two test effort estimation methods. One is the test metric for estimation of software testing efforts to avoid budget overflow, delay in schedule at a very early stage of development. The proposed test metric is a comprehensive one and involves least overhead. The paper also focuses on establishing Cognitive Information Complexity Measure (CICM) as an effective Estimation tool.

Keywords: Testing effort, Requirement based complexity, Cyclomatic number, Function Points

I. Introduction

For a successful software project & proper execution of task, the Estimation Techniques plays key role in SDLC. The method which is used to calculate the time required to complete a particular activity is called Estimation Method. To estimate a task effort different effective Software Estimation Techniques can be used to get a good estimation. Estimation is the process of finding an approximation i.e. a prediction.

An estimate is effort required in terms of

- Time
- Money

An estimate is calculated based on

- Past Experience/Past Data
- Available Documents/Knowledge
- Assumptions
- Calculated Risks

Cognitive complexity based test effort estimation method computes the cognitive complexity of the software and relates it to test effort estimation. It uses line of the code and hence the basic control structure of the program. The method uses a long, tiresome mathematics to at last arrive at a test effort.

Metric based method considers requirements written in standard IEEE format in order to get precise result. Hence the strength of this method lies in finding out requirement based complexity from requirements document before estimating software test effort. This will lead to early error detection which has been shown to be much less expensive than finding defects during implementation or later during testing. The reason for estimation of software testing effort in early phases of SDLC is simple cost. Majority of defects have their root cause in ambiguously defined requirements and also, the cost involved in fixing an error is cheaper if it is found earlier.

II. Background Survey

Effort estimation requires generalization from historical projects. Generalization from limited experience is a constrained problem. In software development, test effort refers to the expenses for tests. Some notable factors which influence test effort are: maturity of the software development process, quality and testability of the test object, test infrastructure, skills of staff members, quality goals and test strategy.

Following approaches are currently used for the estimation: top-down estimation and bottom-up estimation.
Top-down techniques: Function Point Analysis (FPA) and Test Point Analysis (TPA)
Bottom-up techniques: Work Breakdown Structure (WBS) and Wide Band Delphi (WBD).

1) Function Point Analysis: Function points are a measure of the size of project that builds an application. These are based on the use cases. Functional Point is measured from a functional, or user, point of view.
In this FP technique we give a weight to each functional point. Before starting actual estimating tasks functional points are divided into three groups like Complex, Medium & Simple.

- Complex:
- Medium
- Simple

Depending on similar projects & organization standards estimate per function points are defined.

Total Effort Estimate = Total Function Points * Estimate defined per Functional Point
Advantages of the Functional Point Method:

- In pre-project stage the estimates can be prepared.
- Based on requirement specification documents the method's reliability is relatively high.

2) Test Point Analysis: Used for function point analysis for Black box or Acceptance testing. The main components of this method are: Size, Productivity, Strategy, Interfacing, Complexity and Uniformity etc.

3) Software Size Based Estimation:
Let us assume it takes 2 person hours to test software of size one Function Point – using this principle we can easily arrive at the amount of effort required for the project to be tested based on the size of software to be tested. Suppose that the size of software to be tested is 1000 Function points, then, using the norm of 2 person hours per function point, we have 2000 person hours for testing the software of size 1000 Function Points.

4) Test Case Enumeration based Estimation:
The following steps describe this technique –

- List the test cases
- Estimate testing effort required for each test case
- Use Best Case, Normal Case and Worst Case for estimating effort needed for each of the test case
- Compute Expected Effort for each case using Beta Distribution

Best Case + Worst Case + (4 * Normal Case) / 6

Now we sum up the –

1. Expected times to get Expected effort estimate for the project

2. Best-Case times to obtain best-case effort estimate

3. Worst-Case times to obtain worst-case effort estimate

4. Normal-Case times to obtain normal-case effort estimate

5) <u>COCOMO Model:</u> It categorizes software product into three classes-

- Organic Systems - simple projects
- Semidetached Systems- between organic and embedded
- Embedded Systems- complex projects

The model uses a basic regression formula with parameters which are derived from old project data and current as well as future project characteristics. The most important factor in this method is the amount of data that is needed to get regression parameters, which is very rarely in sufficient quantity and thus really limits their usage for project estimation.

III. Issues/Challenges

Disadvantages of Software Testing effort Estimation Techniques:

- hidden factors can be over or under estimated
- Not accurate
- based on thinking
- Involves Risk
- May give false result
- Not reliable
- type of project
- quality requirement goals
- infrastructure and tools needed
- team assembled
- skill of the team members
- project methodology

The techniques discussed in previous section have some challenges related to them. We have discussed those challenges along with the techniques. Few other challenges include:

- Testing size is not computed – therefore, testing productivity cannot be arrived at.
- They cannot be applied to the test-only projects in which the software exists but the testing team does not have access to source code or requirements in order to perform Function Point Analysis or count Source Lines of Code (SLOC).
- Finally, the software size in Function Points or SLOC may not well reflect the amount of test performed by the testing team. This is because there are additional factors affecting test estimation. For example, a stand-alone software application and a web-based software application need a different amount of testing even though their size may be the same.

From the analysis of different techniques we see that each have some benefits while having some challenges. We need to optimize the techniques such that we get a suitable test case estimation technique.

To ensure project/program success, increase overall product quality and improve time to market, it is an imperative that the approach to estimating software test effort is as accurate as possible.

IV. Methodology

A. <u>Cognitive Information Complexity Measure(CICM)</u>

Cognitive Information Complexity Measure (CICM) is an appropriate estimation tool to estimate the testing effort. CICM is an existing software complexity metric that has it foundation on the cognitive aspects of the practitioners. This method is able to demonstrate that the time required to comprehend software is proportional to its cognitive complexity. Weyuker proposed the nine properties to evaluate the software complexity. CICM has also shown its robustness when Weyuker properties were applied on it and has been proven to work for procedural, object-oriented and web-based software. Therefore, it is the proposed approach of software test effort estimation.

- Information in one line of code is the sum of number of all operators and operands in that line. Thus in the kth line of code, information contained is:

Information =f(Identifiers, Operators)

I_k= (Identifiers + Operators) $_k$

= (ID_k + OP_k)

Where IDk = Total number of identifiers in the kth LOC of software and is a positive integer

OPk = Total number of operators in the kth LOC of software and is a positive integer.

Weights are associated with each identifier depending on the type of naming convention used.

- Total Information contained in software (ICS) is the sum of information contained in all lines of code i.e **ICS = $\sum_{k=1}^{LOCS} I$k**
  where Ik = Information content in kth line of code
  LOCS = total line of code in the software.

The information stored in the variables and the necessary operations carried out by the operators in achieving the aim of the software makes the software difficult to understand.

In order to assess the impact of the information contained in a line of code on the number of lines remaining in a program, weighted information count of a line of code is defined next.

- The Weighted Information Count of a line of code (WICL) of software is a function of identifiers, operands and LOC and is defined as:

$WICL_k$ = I_k / [LOCS- k]

Where Ik = Information contained in a software for the kth line.

Therefore Weighted Information Count of the Software (WICS) is defined as:

WICS = $\sum_{k=1}^{LOCS} WICL$k

The measure of complexity should also consider the internal control structures of software as a part of completeness of this measure. Basic control structures of any language logically structure the program control flow and are considered as the Newton’s law in software engineering.

- The sum of the cognitive weights of basic control structures (SBCS) is defined as:

If W1, W2… Wn be the cognitive weights of the basic control structures.

Then **SBCS =** $\sum_{i=1}^{n} Wi$

However, the weights of basic control structures get multiplied in case of nesting since it further increases the comprehension effort. So if there are there are 3 BCS (BCS1, BCS2, BCS3) and BCS 2 is nested in BCS3, then

SBCS = BCS1 + (BCS2*BCS3)

- **CICM = SBCS * WICS**

B. <u>Requirement Based Test Effort Estimation</u>

Practitioners generally evade systematic testing . If we go by quality standards then, the estimation of the software testing effort has to be done before the tester(s) start writing the test cases. Without test effort estimation, the testing manager cannot create a test plan .In order estimate software testing efforts we use improved requirement based complexity measure (IRBC) that is derived from software requirements.

Requirement Based Function Points (RBFP)

IRBC has all the attributes to compute function point analysis (FPA) for any software. In addition to the parameters used by FPA, we considering certain other parameters also for any software requirement. IRBC makes use of an exhaustive set of parameters to compute the Requirement Based Function Point (RBFP) measure. IRBC is capable of computing the complexity value for yet to be developed software while, to estimate RBFP, technical and environmental factors (TEF) pertaining to the testing activity are to be also considered. Degree of influence (DI) that ranges from zero (harmless) to four (essential) determines the need and applicability of these factors. Hence, TEF can be computed by summing the score of nine different factors based on their degree of influence (DI).

Where we set Harmless = 0, Needless = 1, Desirable = 2, Necessary =3 Essential = 4.

Mathematically, TEF can be computed as:

TEF = 0.65 + 0.01 x ΣDI

Thus RBFP can be described as a product of IRBC and TEF, which is sufficient to decide or generate the test case in both black box and white box scenarios. RBFP acts as basis for finding out the optimal number of test cases, required to carry out exhaustive testing.

Number of Requirement Based Test Cases (NRBTC)

Requirement based function point (RBFP) dictate the number of test cases to be designed. Thus we choose Number of requirement based test case (NRBTC) as a function of RBFP expressed as:

NRBTC = (RBFP)1·2

Requirement Based Test Team Productivity (RBTTP)

Requirement Based Test Team Productivity (RBTTP) gives us an estimation of test effort in man hours. Test team productivity depends on the number of staff and personnel (talent) available to test the software, where higher ranked test team will be requiring lesser number of testers.

Requirement Based Test Effort (RBTE)

Prior knowledge of number of test cases and the productivity of the test team is required for testing analysis. We use

NRBTC, for the computation of number of test case and RBTTP, for the estimation of test team productivity. These measures are multiplied in order to get the final requirement based test effort estimate (RBTEE) in man-hours for the proposed software. This is expressed as:

RBTEE = RBTC * RBTTP man-hrs

Early estimation of software testing effort using requirement based complexity will save tremendous amount of time, cost and man power for yet to be developed software.

V. Results

CICM includes all the major parameters that have a relation on the difficulty in understanding software. It clearly establishes a relationship between difficulty in understanding software and its cognitive complexity. the CICM increases with the increase in the Cyclomatic Number for a given program. It introduces a method to measure the amount of information contained in the software thus enabling us to calculate the coding efficiency (EI) as ICS / LOCS. This approach also tells that the test effort increases with the increase in the number of inputs and outputs and the number of BCS's.

It can be deduced from test metric that the test effort estimation is dependent on the software complexity. The unit for IRBC is complexity value; however test efforts are measures in man-hrs. Still a relationship exists between both the parameters, that is, higher complexity requires higher test effort. The values obtained from the proposed approach is comprehensive, well aligned and comparable. Also the test effort computation in man-hrs by the proposed approach is more realistic because it is systematically derived from requirement based complexity (IRBC).

VI. Conclusion

The most important feature of CICM is that it is simple to understand, easy to calculate, less time consuming, gives the complexity value in terms of small number, and language independent i.e. it satisfy most of the property of a good measure. It will aid the developers and practitioners in evaluating the software complexity due to its simple-ness, which serves both as an analyzer and as a predicator in quantitative software engineering.

The test metric measure is reliable because it is derived from SRS of the software to be developed. This makes the estimation precise and perfect. The robustness of the proposed measure is proved by comparing it with different categories of test effort estimations. The proposed work has also been successful in establishing a linear relationship between improved requirement based complexity and requirement based test effort.

VII. References

1. Ashish Sharma, Dharmender Singh Kushwaha, A Metric Suite for Early Estimation of Software Testing Effort using Requirement Engineering Document and its

validation, International Conference on Computer & Communication Technology (ICCCT)-2011

2. Dharmender Singh Kushwaha, A.K.Misra, SoftwareTest Effort Estimation, ACM SIGSOFT Software Engineering Notes, May 2008, Volume 33, Number 3

3. http://en.wikipedia.org/wiki/Test_effort

4. http://www.softwaretestingclass.com/software-estimation-techniques/

5. http://www.softwaretestinghelp.com/software-test-estimation-how-to-estimate-testing-time-accurately/

6. http://www.tutorialspoint.com/software_testing/testing_estimation_techniques.htm

7. http://automation.ultimatetimepass.com/index.php/home/software-testing-estimation-techniques