A Systematic Approach to Determine Bug Severity and Prioritize Fixes

Radha Yadav

Department of Computer Application, Medicaps University, Indore (M.P)

radha.yadav@medicaps.ac.in

Abstract **–** Finding and fixing software's bugs is a not easy task, and a significant amount of effort is dedicated by software developers on this issue. In the world of software one cannot get rid of the bugs, fixes, patches etc. each of them have a severity and priority associated to it. There is not yet any formal relation between these components as both of these either depends on the developer and tester and project manager to be decided on. On one hand, the priority of a component depends on the cost and the efforts associated with it. While on the other, the severity depends on the efforts required to accomplish a particular task. This work proposes a formula that can draw a relationship among severity and priority.

Keywords **–** severity; bugs; priority; test cases; effort estimation; cost estimation;

I. INTRODUCTION

We would try to explore the extent and to derive a reasonable relationship between severity and priority. Currently, we do not a possess a suitable relation between the severity and priority of task to that of severity of it the only component. We know on which these are depended, that are cost associated and the effort to that of severity for the task.

A. Known component

- Estimate effort is the process of predicting the effort required to develop a software system effort based on development, test, and deployment. All these are the level a judgment, based on past experience. But, now past experience depends on a formulating a particular values.
- Cost associated estimate are critical to developer, customer and manager. They can be used for generating request for proposals, contract negotiations, scheduling, monitoring and control.

B Unknown component

- Severity of bug depends on tester or manager, project and their importance. Sometime bug's severity depends on effort and cost of project. It depends on how much; it's severely put impact on project and estimated time to resolve.
- As priority of bug depends on developer or manager, bug's priority depends on effort of tester to find and developer to resolve it and judge, how much it impact on project's smooth running.

However, we know that the effort can be converted to the cost of project task or tests. Also knows the cost that particular project would generate. So, cost can be on of the relating coupling.

II. METHODOLOGY

A Deriving the known components

We can derive the values of the known components by using the following terms:

Efforts required: Effort can be sub-classified into lower granularity as follows [14]:

1) Development Effort
2) Testing Effort
3) Deployment Effort

This can be estimated as:

Effort (E) = DvE+ TeE + DpE

DvE = Development Effort

TE = Test Effort

DpE = Deployment Effort

Calculation of these efforts can be further estimates as: development efforts using PERT and Function point method, these provide more comfort than others method in further calculation of testing and deployment [6]. In case of testing effort, based on test effort work, test case time, test case development & execution time, defect time and use case point approach estimation are used. Deployment effort is based on calculating 10-15% of development effort and by using practical experience of industry person.

1) *Development Effort*

First

PERT estimating i.e. Program Evaluation and Review Technique (PERT) which creates estimates of the weighted average duration of tasks which is given by [13]:

PERT Equation is:

(Optimistic Estimate + (4 times Most Likely Estimate) + Pessimistic Estimate)

Divided by 6

Second

Function Point (FP) based on functionality of a program [1], i.e. the total no. of function point depends on counts of distinct in the five classes:

1) User input types data or control user input types.
2) User output types output data types to the user that leaves the system.
3) Inquiry types interactive requiring a response.
4) Internal file types files (logical group of information) that are used and shared inside the system.
5) External file types files that are passed or shared between the system and other system.

Each of these assigned individual one of the three complexities levels [3]:

Simple = 1, Average = 2 & Complex = 3 & weighting values varies from 3 (simple input) to 15 (complex internal files).

Unadjusted Function Point counts can be given as:

$$UFP = \sum_{i=1}^{5} \sum_{j=1}^{3} N_{ij} W_{ij}$$

N_{ij} W_{ij} are no. and weight of types of class i with complexity.

FP = UPF * CAF

Where CAF is complexity adjustment factor and is equal to

$[0.65 + 0.01 * \sum F_i]$

F_i = (1 to 14) value adjustment factor

Jones's first-order estimation gives an estimate for optimal schedule months from the function point count [2]. First we must choose the appropriate exponent, *j*, to use, by identifying the type of system and the general capability of the development team.

Jones's first order estimate formula uses the exponent, j, from the above table to compute schedule months, s, from function points, f. Schedule months do not include the requirements analysis phase, because this must have been completed to get the design needed for the function point count.

$$\mathbf{s = f^j}$$

Above calculates effort in man-month from function point. The following formula converts function point into total man-months.

$$\mathbf{m=f^{3*j}/27}$$

A spreadsheet could be used to compute this which calculates effort in man-day from function point.

TABLE I. CALCULATE SCHEDULE FROM FUCTION POINT

Kind of Software	Best in Class	Average	Worst in Class
Systems	0.43	0.45	0.48
Business	0.41	0.43	0.46
Shrink-wrap	0.39	0.42	0.45

Value of m divided by 8 (as 8 considered as total working hour per day)

$$\mathbf{DvE = m / 8}$$

2) *Test Effort*

Elements of Test Estimation Process [11]

1) Break sizing into smaller and easier to estimate tasks.
   a. Decompose the test project into phases:
      i. System Test ii. Unit Test
   b. Decompose each phase into constituent activities:
      i. System Test Planning ii. Test Execution
   c. Decompose each activity into tasks and subtasks until each task or subtask at the lowest level of composition:
      i. Executing a test scenario
      ii. Writing a defect

1) Taking risk priority into account
2) Set up dependencies
   a. Dependent tasks internal to the test sub project.
   b. Document dependencies, resources, and tasks external to the test subproject (i.e., those that involve collaborative processes)

- Consider type of code (complex, reused, etc.)
- Augment professional judgment and gut instinct with previous project data, industry metrics, and so forth.
- Identify and, if possible, resolve discrepancies between the test subproject schedule and the project schedule.
- Use the work-breakdown-structure and schedule to develop a budget. Extract from your work-breakdown-structure a complete list of resources.

For each resource, determine the first and last day of assignment to the project.

- If you have resources shared across multiple test projects within a given time period, understand the percentage allocation of each resource's assignment to each project during various time periods.
- Revisit the Estimation continuously in order to reflect any change in the Project Requirements or Schedule
- Be Repeatable preferably Automat

Now, we study various methods helps to calculate the test effort [12]:

First

Total Effort = Test case time + Defect time

Test Case Time = Test Case Development time + Test Case Execution Time
Test Case Development Time = (Hours/Test case development* #Test cases)

Test Case Execution Time = (Hours/Test case Execution * #Test Cases)

Defect Time = (Hours/Defect * # Defects)

Second

Use Case Points Estimation using UCP [Use Case Points], is rapidly gaining a faithful response. The approach for estimation using UCP only needs slight modification in order to be useful to estimate test efforts [10].

1) Determine the number of actors in the system. This will give us the UAW – the unadjusted actor weights. Actors are external to the system and interface with it. Examples are end-users, other programs, data stores etc. Actors come in three types: simple, average and complex. Actor classification for test effort estimation differs from that of development estimation. End users are simple actors. In the context of testing [4], end-user actions can be captured easily using automated tool scripts. Average actors interact with the system through some protocols etc. or they could be Data stores. They qualify as average since the results of test case runs would need to be verified manually by running SQL statements on the store etc. Complex users are separate systems that interact with the SUT through an API. The test cases for these users can only be written at the unit level and involves a significant amount of internal system behavioral knowledge [15].

The sum of these products gives the total unadjusted actor weights. [UAW] as shown in table II below.

2) Determine the number of use cases in the system. Get UUCW.

The use cases are assigned weights depending on the number of transactions / scenarios.

TABLE II. ACTOR WEIGHT

Actor Type	**Description**	**Factor**
Simple	GUI	1
Average	Interactive or protocol-driver Interface	2
Complex	API / low-level Interaction	3

TABLE III. USE CASE WEIGHT

Use Case Type	***Description***	***Factor***
Simple	<=3	1
Average	4-7	2
Complex	>7	3

The sum of these products gives the total unadjusted actor weights. [UAW]

3) UUCP = UAW + UUCW

The calculation of the unadjusted UCP is done by adding the unadjusted actor weight and the unadjusted use case weights determined in the previous steps.

4) Compute technical and environmental factors

The technical and environmental factors for a test project are listed in the table number IV below.

To calculate one needs to assign weights and multiply them with the assigned values to give the final values. The products are all added up to give the TEF multiplier. The TEF multiplier is then used in the next step.

5) Compute adjusted UCP.

We use the same formula as in the UCP method for development.

AUCP =UUCP *[0.65 + (0.01*TEF)]

6) Arrive at final effort.

We now have to simply multiply the adjusted UCP with a conversion factor. This conversion factor denotes the man-hours in test effort required for a language/technology combination. The organization will have to determine the conversion factors for various such combinations.

E.g. Effort = AUCP * 20

Where 20 man-hours and which is divided be 8 for man-day are required to plan, write and execute tests on one UCP.

TABLE IV. TECHNICAL COMPLEXITY FACTOR

Factor	Description	Assigned Value
T1	Test Tools	5
T2	Documented inputs	5
T3	Development Environnent	2
T4	Test Environnent	3
T5	Test-ware reuses	3
T6	Distributed system	4
T7	Performance objective	2
T8	Security Features	4
T9	Complex interfacing	5

3) *Deployment Effort*

First

Based on estimated test efforts as per the industry standard which is taken as 10% of total development efforts (Man Days)

This standard used in many company as well as programming language (application of asp .net & biz talk development).

Second

Basically based on experience, in this process we strongly need at least person whose experience on deployment. Procedure or steps of deployment takes time, each of code copy to execution and acceptance testing to documentation.

DpE= Execution time of application + Installation Time on server (code copy+ code run) + Documentation time + time taken by accepting testing.

After using above all methods we calculate effort, now second thing cost factor that is important for further calculation [6]:

Let the Cost per person hours = CpH

Estimated Cost of Project

$$Ce = E * CpH$$

Now the Cost Generated from the Project be = Cg

Total Revenue of the Project

$$Cr = Cg – Ce$$

Percentage Revenue of the Project = **(Cr * 100)/Cg**

So, we have devised a formula that relates the task to the cost of it [7].

Relating the Known Components to the Unknown Components:

Here, we know that the priority of a task depends on the cost it generates. If there are two tasks, on of which generates higher cost has obvious priority to the other. Also, the severity of a task depends on the components it involves and also the components it impacts

Therefore,

1) Priority ∞ Cost generated (∞ => directly proportional)

2) Severity ∞ No of components Involved + No of components impacted.

Deriving the Constants

Now, to convert the above relation into a formula we would derive few constants. Let's scale the priority and severity on the scale of ten points. We assume the following:

TABLE V. PRIORITY LIST (5 LEVELS]

Priority↓/ Severity→	Low(2)	Medium(5)	High (7)	Critical (10)
A(2)	4	10	14	20
B(4)	8	20	28	40
C(6)	12	30	42	60
D(8)	16	40	56	80
E(10)	20	50	70	100

We have derived the ten point scale by dividing 10 by the number of levels. We get 2 so each class would have a difference of 2.

We can create a constants chart for our reference as follows:

From the below table, we can derive a fair estimate of the severity and priority. Since, we have 1-100 values. We need to derive the cost on the scale of hundred, i.e., the percent value which is known to us. We would assume higher value in as discrepancy about the selection.

Example 1

Follow all the process one be one

Development Effort by PERT

1) At best, need 24 person-hours

2) Most likely need 36 person-hours

3) And if everything goes wrong, you need 51 person hours

Now we convert man per days

36.5/8 = 4.5 person-days = 5 person-days

Test Effort Calculation

Test Case Development Time = 0.16*10 = 16
Test Case Execution Time = 0.083 * 10 = 8.3
Defect Time = 0.16 * 10 = 1.6 hour
Test Case Time = 16 + 8.3 = 24.5 hour
Test Effort work = 24.5 +1.6 **= 26person-hour**

Now we convert man per days

2**6/8= 3.5 person-days = 4 person-days**

Deployment Effort

10-15% of development effort so 1 **person-day**

Total Effort

E = 5 + 4 + 1 = **10 person-days**

Effort per day = 8 hours
Estimated Effort hours: 80 hours
Cost per person hours: 12$
Total Cost: 960 $
Cost Generated from Project: 1200$
Revenue Cost = 1200 – 960 = 240$

The ration of profit to cost generated:

= 240 / 1200 * 100 = 20.

The nearest values are **B-Medium, A-Critical** and **E-Low.**

Now, from the profit generated we know that the number of components is 6. Therefore, the severity is **medium**. Hence, it the severity is medium and priority is **B.**

Example 2

Development Effort by FP

For an **average** case

No of external i/p files -	24	4	= 96
No of external o/p files-	16	5	= 80
No of external inquires-	22	4	= 88
No of internal logical files-	04	10	= 40
No of external interface files-	02	07	= 14

UPF = 318

CAF =. $[0.65 + 0.01 * \sum F_i$ where, $F_{i=}$ (1 to 14)
= [0.65 + 0.01 * (14*3)
= 1.07

FP = 318 * 1.07 = 341

M = $341^{3*0.43}$ (average)
= (341 ^ 1.35)/ 27 = 97.24
Here we can divide no of days 22 or 30
97.24 / 22 = 4.42 = 5 person-days
97.24 / 30 = 3.24 =4 person-days

Test Effort Calculation by UCP

UUCP = UAW + UUCW
= 10+10 = 20
AUCP =UUCP*[0.65+ (0.01*TCE)]
= 20*[0.65+0.01*33]
= 19.6

Effort = 19.6*2
= 39.2 person-hour

Now converting into = 39.2/8
= 5 person-days

Deployment Effort

DpV = 1 hr + 1.5 hr + 2.5 hr + 3 hr
= 8 hr = 1 person-day

Total Effort

E = 4 + 5 + 1 = **10 person-days**

Hence, the severity and priority can be calculated as same in the above quoted example1.

ACKNOWLEDGMENT

We would like to thank the all faculty members of the institute, Prof. Ritesh Shah who helped us lot in calculating the facts and figures related to my paper. I would also like to thank the anonymous reviewers who provided helpful feedback on my manuscript

REFERENCES

[1]. C. R. Symons, "Function Point Analysis: Difficulties and Improvements", IEEE Transactions on Software Engineering, Volume 14, Issue 1, January 1988
[2]. Graham C. Low, D. R. Jeffery, "Function Points in the Estimation and Evaluation of the Software Process", IEEE Transactions on Software Engineering, Volume 16, Issue 1, January 1990
[3]. IEEE Standards Collection: Software Engineering, IEEE Standard
[4]. J. E. Matson, B. E. Barrett, J. M. Mellichamp, "Software Development Cost Estimation Using Function Points", IEEE Transactions on Software Engineering, Volume 20, Issue 4, April 1994
[5]. Hill P.R. (ISBSG) - Software Project Estimation, A Workbook for Macro-Estimation of Software Development Effort and Duration - March, 1999 - Chapter 3

[6]. Johnson K. - Software Size Estimation - Dept. Of Computer Science, University of Calgary, January, 1998
[7]. Londeix B. - Three Points Techniques in Software Project Estimation – SE1ER - April, 1997Inc.
[8]. Capers,John 1996, Applied Software Measurement, Mc Graw-Hill.
[9]. Dekkers Ton, 1999, Test point Analysis Estimation About The Formula.htm
[10]. Cockburn, A Writing Effective Use Cases. Addison Wesley, 2000
[11]. Prof.Torky Sultan, DEVELOPMENT AND EVALUATION OF A DEFECT TRACKING MODEL FOR CLASSIFYING THE INSERTED DEFECT DATA, European Scientific Journal April 2013 edition vol.9, No.12 ISSN: 1857 – 7881
[12]. Aman Kumar Sharma, Comparative Study of the Bug Tracking Tools, International Journal of Advanced Research in Computer Science and Software Engineering, Volume 5, Issue 3, March 2015 ISSN: 2277 128X
[13]. Varun Mittala, Recent Developments in the Field Of Bug Fixing, (ICCC-2014), Bhubaneswar, Odisha, India