Simulation Model for Software Reuse

Kanwaljeet Kaur[1], Er. Vivek Thapar[2]
[1]M-Tech Scholar, Department of CSE, GNDEC Ludhiana, Punjab, India
[2]Assistant Professor, Department of CSE, GNDEC Ludhiana, Punjab, India

ABSTRACT: **Reusability is the likelihood a segment of source code that can be used again to add new functionalities with slight or no modification. Reusable modules and classes reduce implementation time, increase the likelihood that prior testing and use has eliminated bugs and localizes code modifications when a change in implementation is required. Software reuse is the process of implementing or updating software systems using existing software assets. Reuse is the application of existing solution to new problems. In this paper we calculate the efforts of different type of project with reusability and without reusability and we also calculate the mean relative error between original effort and calculated efforts.**

Key Words: **Reusability, Efforts calculation, Simulator, RSC, and Random Number, EAF.**

I. INTRODUCTION

Software reuse has been practiced since programming began. Reuse as a distinct field of study in software engineering, however, is often traced to Doug Mcilroy's paper which proposed basing the software industry on reusable components. Software reuse is generally recognized as a means for reducing the cost of developing an application software system. The process of building the reuse source facility and stocking it with software is called Component Library or domain engineering [1]. It is the process of creating software system from existing software assets rather than building software system from scratch. [1][2] The development of new software from the existing one. The tailoring or alteration of existing software into the new software the concept of software reusing is used, and it can also be used to integrate the innovative concept. The Assets can be software components, software requirement analysis manuals, and design models, database schema, objects, code documentation, domain architecture, standards, test scenarios, and plans. The existing software can be from within a software system or other similar software systems or widely in different systems.

Software reuse can occur at many levels, ranging from the reuse of small granules of function (small software objects) within an application system to the reuse of large granules (large software objects) of software function across many application systems.

The principal economic benefits of software reuse are: [3]

- Lower development costs.
- Higher software product quality due to multiple testing and error removal opportunities reused over a number of application systems.
- Reduced development schedule due to a reduced amount of development work.
- Lower maintenance costs due to lower levels of error creation.
- Reduced life cycle costs due to reduced development and maintenance costs.

Software reuse is an important approach to increase software quality and productivity. In a software reuse procedure, extracting the software components with high reuse potential and high quality is a key step that may directly affect the steps that follow. Some software metrics can measure the quality characteristics of the RSC (Reusable Software Component). However, individual software metric cannot measure the overall quality characteristic of the RSC. Therefore, the software metrics shall be combined, and the conflict situations in metric combination should be reduced.

Rest of the article is organised as. Firstly we introduce concept of software reuse, then we elaborate reuse strategies. After that we introduce our simulation work with proposed model. At the end we conclude with results and references.

II. REUSE STRATEGIES

No matter which approach is selected, or what size the reusable software artifacts it deals with, [3][4]it is vital-to have a properly developed reuse strategy if any benefits are to be gained from reuse. It has been suggested that a reuse strategy must include features to support the following technical aspects.

- **Abstraction** - The ability to provide succinct, high level descriptions of reusable artifacts is essential in assisting the developers in understanding their purpose, nature and behaviour.
- **Selection** - To aid a developer in performing reuse, mechanisms that allow the location, comparison and selection of artifacts are also important. In effect, these require directory and search services.
- **Specialisation** - Any technique that maintains collections of reusable abstract artifacts requires features to support

the specialisation of those artifacts into useable, concrete entities, since it is unlikely that large grained components can be reused without modification.

- **Integration** - An integration 'framework' is required to provide a mechanism for the combination, connection and communication between reuse artifacts.

This must include an agreed architecture within which reuse is achieved.

III. SIMULATION

Our simulator is implemented on high level language "C Programming Language"**.** C provides functionality to implement applications. C allows building large programs with clarity, extensibility and ease of maintenance. In C there are many header files which provide larger functionality.

Simulation is the process of designing a model of a real system and conducting experiments with this model for the purpose either of understanding the behavior of the system or evaluating various strategies within the limits imposed by a criterion or a set of criteria for the operation of the system. Once a simulation is in the use, running it on new data or with new parameters is usually just a matter of few keystrokes or dragging and dropping a different life. Depending upon the variables being deterministic or random, the simulation models can be classified as:

A. Deterministic Simulation
B. Stochastic Simulation

In a deterministic simulation, a system is simulated under well determined conditions. This kind of simulation is useful to observe the behavior of system in certain particular cases, to discover errors in the design or in the implementations, to build examples, etc. In this kind of simulations, only one run is needed and there is no truly random variable involved.

A system that relies heavily upon random behavior is referred to as a stochastic system. In Stochastic simulation, system performance is measured. This is useful to see if the system has good response time under average conditions, to compare different implementations of the same system, or totally different systems that have the same output. It is useful to classify the system being simulated into separate categories depending upon the degree of randomness associated with behavior of the system in its simulated environment.

Our simulator is working for three types of projects, categorised under COCOMO model

- Organic
- Semidetached
- Embedded

These categories roughly characterize the complexity of the project with organic projects being those that are relatively straight forward and developed by a small team, and embedded are those that are ambitious and high requirements for such aspects as interfacing and reliability.

The values of constant a and b in different types of Projects are as:

Projects	A	B
Organic	3.2	1.05
Semi detached	3.0	1.12
Embedded	2.8	1.20

Table 1: Constants value for Different Projects

IV. PROPOSED WORK

A software component or software is available for modification to make it reusable and will require time for change. The simulator will calculate the time of completion of the project which is using reusable software over several simulation runs. This will give clear idea about the service for the project. Reuse managers can determine the service time for the project and completion time for project. The simulator will calculate the effort for the project over several simulation runs. After calculated efforts it provides the clear idea to reuse managers how to allocate resources.

For cost driver factors there can be different scale.
The rating scale for RELY can be very-low, low, high, very high. These values are randomly generated. Ex 0.75, 0.88, 1.00, 1.5.

To calculate EAF (Effort Adjustment Factor) multiply all these cost drive attributes. EAF can be different for different projects and there is possibility that rating scale for cost driver attributes can be changed due to random generation of numbers.

According to standardized normal distribution function and generating random numbers from the standardized normal distribution function the following relation called the Box-Muller transformation as:

$$\mathbf{S= ((-2 * \log_e r1)^{1/2} *\cos(2* \pi * r2))}$$

Where r1 and r2 are two uniform random numbers in the range (0.1) and s is the desired sample from the standardized normal distribution.
Then we calculate relative error:

Relative error = ((calculated effort-original effort) / (original effort)))*100

In order to implement our proposed model, we developed an Application which identifying importance of software reuse in cost estimation. It uses two functions namely:

a) EFFORT (), to calculate efforts for the different type of projects.
b) PRODUCE (), to generate random samples of cost driver attributes.

Project no.	Size (KLOC)	Original effort
1	50	47
2	40	66
3	22	60
4	13	159

Table 2: Input data for the project

Input value of μ =0.567,σ =0.007

V. RESULTS

The following results are calculating efforts and relative errors without reusability.

For Organic Project without reusability the efforts are as:

Project no	Size (KLOC)	Original Effort	Calculated effort	Relative Error%
1	50	47	194	134
2	40	66	153	32
3	22	60	82	-22
4	13	159	47	-83

Table 3: Organic Project

For Semi Detached Projects without reusability:

Project no	Size (KLOC)	Original Effort	Calculated effort	Relative Error%
1	50	47	239	269
2	40	66	186	101
3	22	60	95	8
4	13	159	60	-78

Table 4: Semi Detached

For Embedded Projects without reusability:

Project no	Size (KLOC)	Original Effort	Calculated effort	Relative Error%
1	50	47	306	189
2	40	66	234	60
3	22	60	114	-9
4	13	159	53	-81

Table 5: Embedded Projects

The following results are calculating efforts and relative errors with reusability on the same projects.

Calculation of size when we are using the existing software in development of new software is as:

Effective size= new code + size of existing software* (0.4*25% +.25 *10% + 0.35 * 18%)

For Organic Project with reusability the efforts:

Project no	Old Size	New Size(KLOC)	Original Effort	Calculated effort	Relative Error %
1	50	42	47	162	95
2	40	34	66	134	11
3	22	20	60	74	-29
4	13	13	159	47	-83

Table 6: Organic Project

For Semi Detached Projects with reusability:

Project no	Old Size	New Size(KLOC)	Original Effort	Calculated effort	Relative Error %
1	50	42	47	198	138
2	40	34	66	156	33
3	22	20	60	86	-18
4	13	13	159	53	-81

Table 7: Semi Detached
For Embedded Projects with reusability:

Project no	Old Size	New Size(KLOC)	Original Effort	Calculated effort	Relative Error %
1	50	42	47	249	199
2	40	34	66	193	65
3	22	20	60	102	-3
4	13	13	159	60	-78

Table 8: Embedded Projects

VI. CONCLUSION

We have developed simulator which calculates efforts and effective size for the projects with reusability and without reusability by taking the values of a and b for various projects. In addition simulator also calculates cost driver attributes by Box mullar Transformation after generating random numbers. After 1000 of simulation runs it calculates the efforts and relative error between original and calculated efforts.

REFERENCES

1. Sarbjeet Singh, Sukhvinder Singh, Gurpreet Singh **"Reusability of the Software"** International Journal of Computer Applications (0975 – 8887) Volume 7– No.14, October 2010.
2. G. N. K. Suresh Babu And Dr. S. K. Srivatsa **"Analysis And Measures Of Software Reusability"** International Journal of Reviews in Computing, E-ISSN: 2076-3336.
3. J. E. Gaffney, Jr. and R. D. Cruickshank **"A General Economics Model of Software Reuse"**.
4. Michael Pidd **"Simulation Software And Model Reuse: A Polemic"** Proceedings of the 2002 Winter Simulation Conference E. Yiicesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, eds.
5. Martin Griss. **Software Reuse: Objects and Frameworks are not enough**. 1995.
6. Zhengxin Chen. **On Analogy For Software Reuse : A Perspective From Cybernetics**. 1995.
7. Jean-Marie Burkhard and Francoise Detiennie. **An Empirical Study of Software Reuse By Experts in Object-Oriented Design** 1995.
8. Kimberly Jordan. **Software Reuse Term Paper**. 1997.
9. Mark Lattanzi, Sallie Henry. **Software Reuse Using C++ Classes: The question Inheritance.** 1997.
10. Dan Galorath. **Software Reuse and Commercial Off-the-Shelf Software**. 1997.
11. J. M. Perry. **Perspective on Software Reuse. Technical Report** CMU/SEI- 8TR022 ESD-TR-88-023. 1988.
12. R. K. Raj, H. M. Levy. **A Compositional Model for Software Reuse**. In the computer Journal, vol. 32, No. 1989.
13. W.Tracz. Tutorial: **Software Reuse: Emerging Technology**. In IEEE Computer Society Press, Los Alamitos, Calif.1988.
14. Norman Fenton and Austin Melton**, Driving Structurally Based Software Measures**. J. System Software,(12) 1990, pp. 177-187.
15. Martin L. Griss Software and Systems Laboratory **"Software Reuse at Hewlett-Packard"** March, 1991.