A Metric Extent for Component based Software Products

Kumar Rahul
Email: hitechsmart@gmail.com
Research Schoar, Mewar University,
Chittorgarh(Rajasthan)

Dr B K Sinha
Email: brijeshsinha@gmail.com
Dean(Engineering), Rama Engineering College
Ghaziabad(UP)

***Abstract*—**The overriding goal of software engineering is to provide a high quality system, application or a product. To achieve this goal, software engineers must apply effective methods coupled with modern tools within the context of a mature software process [2]. In addition, it is also must to assure that high quality is realized. Although many quality measures can be collected at the project levels, the important measures are errors and defects. Deriving a quality measure for reusable components has proven to be challenging task now a days. The results obtained from the study are based on the empirical evidence of reuse practices, as emerged from the analysis of industrial projects. Both large and small companies, working in a variety of business domains, and using object-oriented and procedural development approaches contributed towards this study. This paper proposes a quality metric that provides benefit at both project and process level, namely defect removal efficiency (DRE).

***Keywords*—**Software Reuse, Defect density, Reuse metrics, Defect Removal efficiency.

I. Introduction

OVER the past ten years, the software reuse and software engineering communities have come to better understanding on component-based software engineering. A good software reuse process facilitates the increase of productivity, quality, and reliability, and the decrease of costs and implementation time [3]. An initial investment is required to start a software reuse process, but that investment pays for itself in a few reuses. In short, the development of a reuse process and repository produces a base of knowledge that improves in quality after every reuse, minimizing the amount of development work required for future projects, and ultimately reducing the risk of new projects that are based on repository knowledge [1]. In this context, defect removal efficiency can be used as quality metric when developing a software product.

A. Software Reuse

Software reuse is the process of implementing or updating software systems using existing software assets. Software assets, or components, include all software products, from requirements and proposals, to specifications and designs, to user manuals and test suites [3]. Anything that is produced from a software development effort can potentially be reused. Reuse can be achieved through different modes. Compositional reuse involves constructing new software products by assembling existing reusable assets, while generative reuse involves the use of application generators to build new applications from high level descriptions [4].

B. Present State in the Reuse World

In industry, information on defect density of a product tends to become available too late in the software development process to affordably guide corrective actions. An important step towards remediation of the problem associated with this late information lies in the ability to provide an early estimation of defect density [7]. In the current scenario, testing is the key method for dynamic verification and validation of a system. Any deviation from the system's expected function is usually called as a failure and these failures are communicated to the developers by means of failure repots. The terms error is used both for execution of a "passive" fault leading to erroneous behavior or system state [6], or for any fault or failure that is a consequence of human activity [5]. Some times the term defect is used instead of faults, errors or failures, not distinguishing between active or passive faults or human/machine origin of these. In this paper, we are proposing a quality metric for the estimation of defects.

C. Productivity Benefits on Reuse

Reuse has been advocated as a means for reducing development cost. For example reuse of components is identified as one of the most attractive strategies for improving productivity. It improves productivity, because the life cycle now requires less input to obtain the same output or productivity may increase simply because fewer work products are created from scratch. In general, reuse improves productivity by reducing the amount of time and labor needed to develop and maintain a software product. Based on data presented in [3], we can conclude that there is strong relationship between productivity and reuse rate, which is shown in Fig. 1.

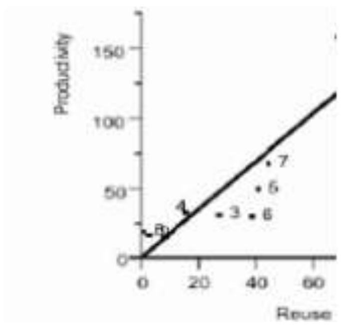


Fig. 1 Relationship between productivity and Reuse

D. Quality Benefits on Reuse

The accumulated defect fixes result in a higher quality work product because work products are used multiple times, Moreover, Reuse prevents and removes defects earlier in the life cycle because the cost of prevention and debugging defects can be amortized over a greater number of uses [8]. According to the data presented in [3], there is a strong linear relationship between reuse rate and project error density (quality), which is shown in Fig. 2.

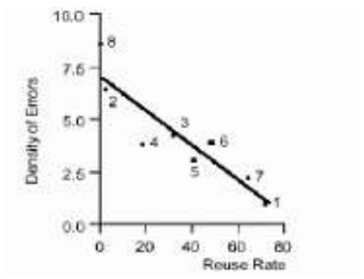


Fig. 2 Relationship between Quality (error density) and Reuse

E. Defect Removal Efficiency (DRE)

DRE is a measure of the filtering ability of quality assurance and control activities as they are applied throughout all process framework activities. The computation of DRE can be done as follows [2].

$$DRE=E / (E+D)$$

where E is the number of errors found before delivery of software to the end-user. D is the number of defects found after delivery.

The ideal value of DRE is 1. As E increases for a given value of D, the overall value of DRE begins to approach 1.

We can also define DRE as $DRE_i = E_i / (E_i + E_{i+1})$ Where E_i is the number of errors found during ith software engineering activity and E_{i+1} is the number of errors found during the software engineering activity i+1 those were not discovered in the activity i. ie, The errors that are not found during the review of the analysis phase are passed on to the design phase. Fig. 3 shows the defect removal process during software development life cycle (SDLC).

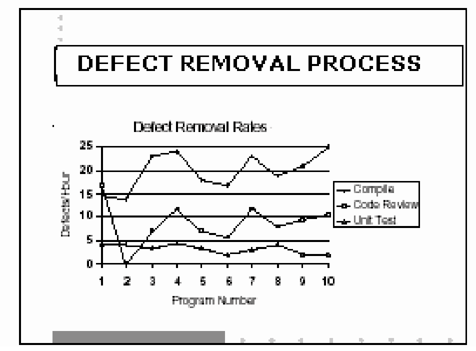


Fig. 3 Defect Removal Process During SDLC

F. Redefinition of DRE in the Reuse Context

Classifying and counting defects helps focus problem solving and root cause analysis efforts. Historical defect data can assist organizations early in the project planning process to predict defect insertion and find defect counts for each component integration stage .It is also important to recognize the rate at which defects accumulate.

Defect insertion and removal process over the course of a reuse based development scenario are summarized in Fig. 4. The left curve illustrates that defect insertion (in the form of inappropriate match, requirement compromise etc.) begins when the project effort begins. The second curve illustrates that finding and fixing defects most often occurs substantially after the initial component testing.

In the incremental approach of Integration or When COTs are used for software development, we can redefine DRE as

$$DRE_i = E_i / (E_i + E_{i+1})$$

where E_i is the number of errors found in the ith component and E_{i+1} is the number of errors found after integrating i+1th component with the ith component. In this context, to obtain the high quality product, DRE for the whole product should approach 1. That means finding all errors of each component ensures the ideal value for DRE.

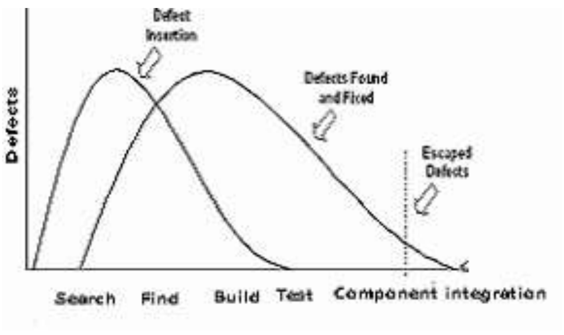


Fig. 4 Defect insertion and find and fix Scenario

II. IMPLEMENTATION

A. Data Set Required for DRE Computation

No. of errors found in component 1(c1)

No. of errors found in component 2(c2)

…………………………………….

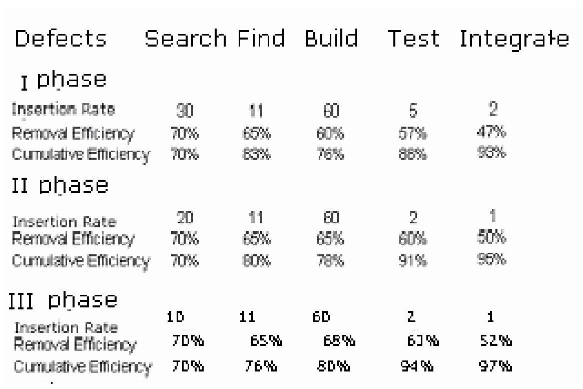
No. of errors found in component n (cn)

Compute DRE $_i$ = E $_i$ / (E $_i$ + E $_{i+1}$) before each integration.

B. *Planning a Defect Profile*

For each stage of the COTs based Software development, the following steps can be adopted.

Steps Involved:
Step1.Estimate the number of defects likely to be inserted (n)
Step2.Estimate the removal efficiency (y as %)
Step3.Calculate the number of defects likely to be removed (y*n)
Step4.Calculate number remaining (n-y*n)
Step5.Add to estimate of the number likely to be added in next stage of component integration
Step6.Calculate cumulative removal efficiency (as %)



Defects	Search	Find	Build	Test	Integrate
I phase					
Insertion Rate	30	11	60	5	2
Removal Efficiency	70%	65%	60%	57%	47%
Cumulative Efficiency	70%	83%	76%	88%	93%
II phase					
Insertion Rate	20	11	60	2	1
Removal Efficiency	70%	65%	65%	60%	50%
Cumulative Efficiency	70%	80%	78%	91%	95%
III phase					
Insertion Rate	10	11	60	2	1
Removal Efficiency	70%	65%	68%	63%	52%
Cumulative Efficiency	70%	76%	80%	94%	97%

Fig. 5 Defect insertion Rate and Removal effiency

The Fig. 5 shows the different stages of defect insertion rate, removal efficiency and cumulative efficiency in the component based software development process. In the figure only three phases of integration are shown. The process will continue till the final product development completion. After this process, there is a defect-reporting phase. This phase consists of following steps.

C. *Defect Reporting Phase*

Step 1. Defect Log (Where found, date found, type, stage injected, stage removed, consequences of removal, time to repair, etc)
Step2. Defect report forms (Location, severity, inspection rates, yields, etc.)

III. Conclusion

The success in development, maintenance and continued improvement of the systems has been achieved by a careful reuse of components. The reuse orientation provides many advantages, but it also requires systematic approach in design, planning, extensive development, support of a more complex maintenance process, and in general, more consideration being given to quality (error density) of components. The more a reusable component is developed, the more complex is the development process and more support is required from the organization to ensure the quality of the developed product. This paper redefines the basic definition of defect removal efficiency in terms of the phases involved in the reuse based development and also gives a systematic approach in the defect removal process.

Acknowledgment

This study has been possible thanks to the support and collaboration of several industry people they are directly or indirectly contributed towards the successful completion of this research.

References

[1] Kimberly Jordan, MJY Team, George Mason University, "Software Reuse Term Paper For The MJY Team", Software Risk Management WWW SITE, Apr.1997.

[2] Roger S Pressman, *Software Engineering-A practitioner's approach*, 5th Edition, McGraw-Hill, 2001.

[3] V. Basili, L. Briand, and W. Melo., "Measuring the impact of reuse on quality and productivity in object-oriented systems." Technical Report CS- TR-3395, University of Maryland, Computer Science Department, 1995.

[4] Barnes, B. H., Bollinger, T. B., "Making Reuse Cost-Effective," IEEE Software, Vol. 8, Number 1, January 1991, pp. 642-652.

[5] J. E. Gaffney, Jr., R. D. Cruickshank, "A general economics model of software reuse", Proceedings of the 14th international conference on Software engineering, Melbourne, Australia, May 11-15, 1992, pp.327-337.

[6] Fonash P., " Metrics For Reusable Software Code Components", PhD Dissertation, George mason University, Fairfax, Virginia, 1993.

[7] Parastoo Mohagheghi, Reidar Conradi, Ole M. Killi, Henrik Schwarz, "An Empirical Study of Software Reuse vs. Defect- Density and Stability", Simula Research Laboratory, P.O.Box 134, NO-1325 Lysaker, Norway.

[8] Lubars MD, *Affording Higher Reliability Through Software Reusability*, Software Eng. Notes, Oct. 1986.

[9] Fenton, N.E., Ohlsson, N., "Quantitative Analysis of Faults and Failures in a Complex Software System", IEEE Trans. Software Engineering, 26(8), 2000, pp. 797-814.

[10] Malaiya, K.Y., Denton, J., "Module Size Distribution and Defect Density", Proc. 11th International Symposium on Software Reliability Engineering- ISSRE'00, 2000, pp. 62-71.

[11] Fonash P., "Metrics For Reusable Software Code Components", PhD Dissertation, George mason University, Fairfax, Virginia, 1993.

[12] Sherriff, M., Nagappan, N., Williams, L., and Vouk, M. A., "Early Estimation of Defect Density Using an In-Process Haskell Metrics Model," First International Workshop on Advances in Model-Based Software Testing, St. Louis, MO, May 15-21, 2005.