International Journal of Computer Theory and Engineering, Vol. 1, No. 4, October2009
1793-8201

Reengineering Test Management for Increasing Testing Effectiveness in Component Based Enterprise Applications

Jasmine K.S and Dr. R. Vasantha

***Abstract*—Software reuse is widely considered as a way to increase the productivity and improve the quality and reliability of new software systems. Identifying, extracting and reengineering software components, which implement abstractions within existing systems is a promising cost-effective way to create reusable test assets. In the present scenario, one of the major problems in building large-scale enterprise systems is anticipating the performance of the eventual solution before it has been built. Testing is an important and significant part of the software development lifecycle to ensure a high quality product with a minimum number of faults. But most organizations don't have a standard process for defining, organizing, managing, and documenting their testing efforts. Often testing is conducted as an ad hoc activity, and it changes with every new product. Without a standard foundation for test planning, development, execution, and defect tracking, testing efforts are nonrepeatable, nonreusable, and difficult to measure. Reengineering the test management process can solve the problems due to unstructured, decentralized test management. This paper explains the goals of reengineering test management and how to achieve it and the approach as demonstrated, constructs useful models that act as predictors of testing effectiveness in component based enterprise applications.**

***Index Terms*—Software Reuse, Component-based development, Test management, Software components, Reengineering.**

Ⅰ INTRODUCTION

In recent years, two accomplishments have fueled an upsurge in the complexity of scientific simulation software. First, rapid growth in computational capability based on increasingly intricate hardware architectures is driving computational scientists to develop new, more complex algorithms to make best use of the systems. Second, scientific advances are yielding new ways of approaching challenging problems, offering better efficiency, accuracy, or fidelity.

Code complexity and reliance on software are increasing as essential consequences of both of these accomplishments. Computational science software is at growing risk of becoming a victim of its own success, increasing in complexity until it becomes unmanageable, unmaintainable and incomprehensible. This inherent complexity impacts the productivity of developers and, if left alone, ultimately may cap the rate of progress in creating and improving scientific software.

Jasmine K.S is with RVCE, Visveswaraya Technological University, Asst.Professor, Dept of MCA, Bangalore, India (phone: +919916101571).

Dr. R. Vasantha is with RVCE, Visveswaraya Technological University, Prof, Dept of ISE, Bangalore, India.

Component-based software engineering (CBSE) is an approach developed in other areas of computing as a means of addressing similar problems of complexity. Units of software functionality are encapsulated as components, which interact with each other only through well-defined interfaces. The actual implementation is opaque to other components, and application composition is archived through providing and using these interfaces. This approach reduces complexity by allowing developers to focus on the internals of the small set of components on which they are working, while users of components need only be concerned with component interfaces [7]. This separation of concerns is useful in the collaborative or community-oriented software development that increasingly characterizes modern high-end simulations. Component-based environments typically offer a "plug and play" approach to composition of components into applications, in which components offering the same interface are interchangeable, allowing easy swapping of components to test new algorithms, tune for performance, and other reasons [4]. To the extent that communities of users agree to interfaces for certain functionality, components can more easily be reused across multiple applications. Reengineering allows altering an existing system to reconstitute it in a new form.

A. Reengineering and Reuse

Chikofsky and Cross define reengineering as "the examination or alteration of a subject system to reconstitute it in a new form and subsequent implementation of that form" [14]. This definition clearly is focused on the typical interpretation of the term, the alteration of a software artifact. Arnold, on the other hand, defines reengineering as "any activity that (1) improves one's understanding of software, or (2) prepares or improves the software itself, usually for increased maintainability, reusability, or evolvability"[6]. The term interpretation is particularly salient in organizations that have lost the key individuals in which the knowledge regarding the system of interest resides. There are a number of potential benefits derived from reengineering software. Reengineering can help reduce an organization's evolution risk. It is not uncommon for the only source of information regarding why a software system does what it does to be the software system itself. Reengineering hence can help an organization recoup its investment in software and retain its corporate memory. Reengineering can make software easier to change and improve its reusability. Incorporation of new design and implementation techniques can create a more modular and composable system, accommodating not only future modifications more effectively but also the recomposition of a system into a new configuration for a new

application via techniques such as refactoring [5]. Hence reengineering is a catalyst for automating software maintenance, providing a cusp through which an organization can take a fresh direction in its development and support of the organization's software portfolio, as well as acting as a potential agent for enabling reuse within the organization.

Fig1. Reengineering

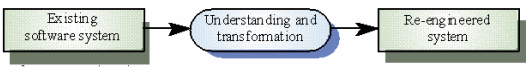

Fig2. Forward engineering

B. Component-Based Software Reengineering

The component based software Re-engineering is aiming to achieve larger software reutilization during software systems reconstructing, the use of Component-Based Software Engineering (CBSE) techniques in reengineering is being researched [20]. The combination of Reengineering and CBSE techniques can be a solution to rebuilt existing systems, reusing the knowledge that has been accumulated during their usage over the years and delivering more evolvable and maintainable systems.

There are two types of reengineering can be considered [18]:

i) White-Box Reengineering

ii)Black-Box Reengineering

In White-box reengineering, the repartitioning of the existing system into a component-based system takes place. It requires an in-depth understanding of existing systems from the bottom-up. It performs low-level changes to the source code.

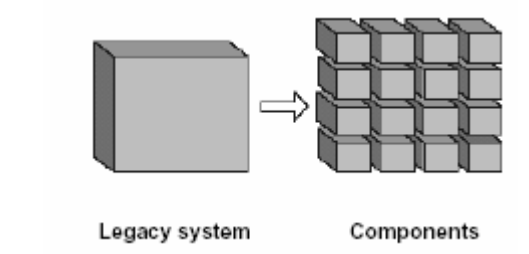

Fig3. White-box reengineering

In the black box reengineering, addition of a new component layer to the existing system without change to the underlying source code takes place. In this approach, one big black box breaks into a number of smaller conceptual black boxes, each representing a high-level business component. It Presents the existing system to the outside world as though it were constructed from a number of software components.

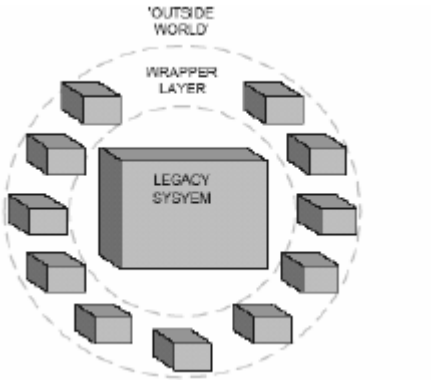

Fig4. Black box reengineering

C. The Need to Reengineer Test Management

Testing is such an important and significant part of the software development lifecycle. Most organizations don't have a standard process for defining, organizing, managing, and documenting their testing efforts. Often testing is conducted as an ad hoc activity, and it changes with every new project. Without a standard foundation for test planning, development, execution, and defect tracking, testing efforts are nonrepeatable, nonreusable, and difficult to measure. Generating a test status report is very time consuming and many times not reliable. It is difficult to procure testing information such as the quantity of testing process, completion time of testing, results of tests conducted, sharing of the test cases in a similar kind of a platform and the level of test coverage of the requirements specified. Getting this information fast is critical for software product and process quality. But many times, it is difficult to get this information, depending on the way test cases and execution results are defined, organized, and managed. There are many problems associated with defining and storing test cases in decentralized documents such as treacking, reuse, duplication of test cases and efforts, version control, changes and maintenance, inconsistenrt processes, requirement tracebeality and coverage. The reengineering of test management system facilitates to organize the material from various reuse activities so they can be shared effectively for use across the organization.

The objectives of the reengineering testing approach can be listed as follows:

- Assist in defining, managing, maintaining, and archiving test ware
- Assist in test execution and maintaining the results log over different test runs and builds
- Centralize all testing documentation, information, and access
- Enable test case reuse
- Provide detailed and summarized information about the testing status for decision support
- Improve tester productivity
- Track test cases and their relationship with requirements and product defects

Ⅱ IMPLEMENTATION

A. Proposed System

Steps involved

1. Perform reverse engineering to understand the existing system
2. Perform the transformation trough translation of the change scenario required
3. Achieve the Re-engineered system

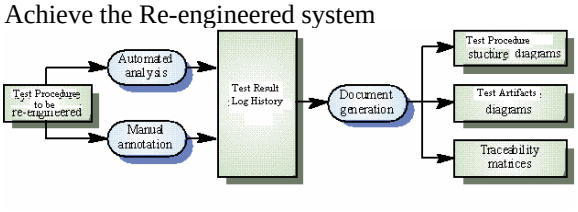

Fig5. The Reverse Engineering process

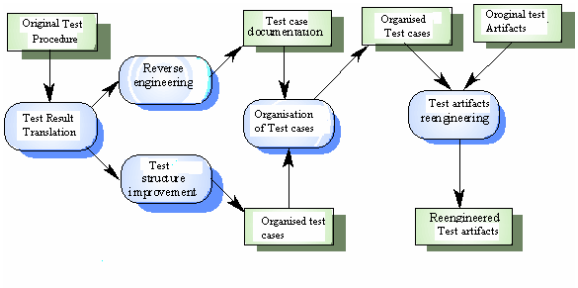

Fig6. Reengineering process

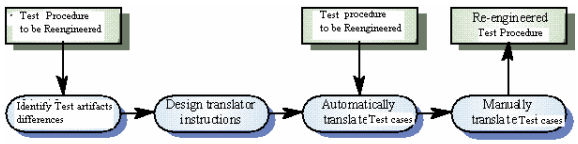

Fig7. The Test procedure translation process

The step, reverse engineering, is used to rebuild the system representation at the test design level. From the test results log histoty, test procedure structure diagrams and test artifacts design diagrams are generated and documented to describe and control testing hierarchies. These diagrams combined with existing test case documentation, personnel experience and domain knowledge make it possible to reproduce all the information required at the test design abstraction level. A number of software technologies are today available; including traditional CASE tools themselves, which incorporate reverse engineering capabilities. Traceability matrices generated in this phase is useful to check the mapping of the design changes with the requirements. Next, the test structure improvement will take place with the help of test case organization and test case documentation. Then test artifacts will be reengineered based on the organized test cases. In the translation phase, identification of test case differences in the component level and the necessary test structure changes in the interface will takes place. Old test design documents are compared with the new rebuilt test design documents to verify that design changes have not resulted in a worsening of test results. The last step, forward engineering, is performed to upgrade those parts of the systems, which have been redesigned, and to ensure that the system still works with a better quality.

B. Cost Considerations

In the industry world, different reuse scenarios are observed, namely i) Systematic reuse (SR) ii) Controlled reuse (CR) iii) Opportunistic reuse (OR) and iv) Pure development (PD). A reuse scenario is any sequence of elementary operations performed while practicing reuse. In the Systematic reuse scenario, the ultimate goal of reuse processes is to have a set of test assets that are readily available for reuse in all future products without further modification. In the Controlled reuse scenario, a core-asset repository has been established in which test assets are stored or cataloged for the benefit of other products. In the Opportunistic reuse scenario, each group responsible for one of the n products knows that there exists a viable source S, but it is not stored in any shared repository or registered in a public catalog. In the Pure development scenario, each group responsible for one of the n products is unaware of the existence of S and, therefore, develops its target component from scratch.

In order to determine the optimal scenario through which the final target can be obtained from the original source test assets, we must be able to compare the relative cost of alternative scenarios. The basic operations in a reuse scenario are

1) Mining and Cataloging (MC): Identifying and acquiring an existing private asset P, from a certain product, and then storing and cataloging it formally as a repository asset R.The associated cost is the total cost of domain analysis.

2) Copy and Paste (CP): Acquiring a copy of a private asset P for a specific product. The source asset is not cataloged in the repository, and awareness of its existence is based on personal knowledge. The cost of CP is identical to cost of MC, i.e, the cost of domain analysis of a single asset.

3) Black-Box reuse (BB): Acquiring a copy of a particular repository asset with no modifications for a specific product as a private asset. Ideally, this should be an elementary copy operation; in practice, however, this operation may require some overhead activities as a consequence of adapting the architecture of the target product in order for the imported asset to fit. This is also the case when acquiring COTS test assets for the product.

4) Cataloged asset Acquisition (CA): Acquiring a copy of a repository asset R for a specific product as a private asset P. It is assumed that P needs to undergo further modifications (white-box reuse) within the product in contrast to black box reuse. This cost includes the effort invested (or that may be invested) in searching for an appropriate asset in the repository, then analyzing and evaluating its fit with the target product. There can be acquisition of asset from external source and cataloging it as a repository asset R.This is the case with COTS artifacts.

5) White-Box reuse (WB). Modifying an existing private asset P into another private asset P1 within the same product. This is the average cost of learning the asset's structure plus asset modification and adaptation over all the reuses of this asset in the pilot study.

6) Number of Reuses. The number of target products that reused the asset during the pilot study.

TABLE I: THE COST OF BASIC OPERATIONS- A SAMPLE DATA [25]

Basic operation \ Asset	1	2	3	4	5	6	7
Mining and Cataloging (MC) [1]	**110**	**110**	**110**	**110**	**110**	**110**	**110**
Copy and Paste (CP) [2]	110	110	110	110	110	110	110
Black-Box reuse (BB) [3]	**100**	**200**	**150**	**100**	30	400	**10**
Catalog Acquisition (CA) [4]	100	100	50	50	**20**	**50**	50
White-Box reuse (WB) [5]	300	300	400	300	**50**	**300**	700
Number of reuses [6]	2	3	3	2	3	3	4

The table 1 describes the cost of various basic operations in a reuse scenario for a sample of seven test assets.

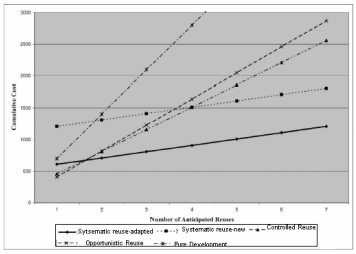

Fig8. The cost of different reuse test scenarios

Figure8 depicts the cost-effectiveness of the actual test reuse scenarios implemented for each of the test assets relative to other scenarios: Opportunistic and Pure Development. Systematic reuse-adapted represents the core developed from scratch [18]. test assets adapted from test artifacts mined during domain analysis and Systematic reuse –new represents the core test assets

TABLE II: RELATIVE SAVINGS OF ALTERNATIVE REUSE SCENARIOS- A SAMPLE DATA [25]

Ratio \ Asset	1	2	3	4	5	6	7
Systematic Reuse vs.							
Controlled Reuse	**11%**	**8%**	**34%**	**12%**			**63%**
Opportunistic Reuse	**1%**	**2%**	**37%**	**13%**			**65%**
Pure Development	**42%**	**73%**	**54%**	**49%**			**81%**
Controlled Reuse vs.							
Systematic Reuse					**-28%**	**32%**	
Opportunistic Reuse					**33%**	**6%**	
Pure Development					**41%**	**74%**	
Number of reuses	2	3	3	2	3	3	4

Table 2 presents the cost-effectiveness of the actual reuse scenarios implemented for each of the test assets relative to other scenarios: Opportunistic Reuse, and Pure Development We can see from Table 2 that if, the choice were between only Systematic Reuse and Controlled Reuse, then the largest relative savings (63 percent) would have resulted from the Systematic Reuse of Asset 7. Conversely, the worst choice would have been the Controlled Reuse of Asset 5 at a cost of 28 percent relative to the alternative. In comparison the other referenced scenarios, we can see that the relative savings obtained by implementing the preferred scenario over Opportunistic Reuse were between 1 and 65 percent. In comparison to Pure Development, the relative savings were even more dramatic—between 41 and 81 percent.

III CONCLUSIONS & FUTURE STUDY

The reengineering and reuse of large legacy software systems can be an expensive, error-prone endeavor. Organizations are increasingly recognizing their software portfolios as test assets to be utilized rather than as obsolescent artifacts that should be discarded at the first opportunity [17]. However, this shift in perception is not without a price. Interest in software representation as a major factor in the software development cycle has steadily been gaining in importance. This leads to an evolving perspective of a spectrum of representation expressiveness in software development environments and tools [6]. This paper surveys some of the key issues in reengineering legacy software systems and the adoption of reuse and reengineering technology in test management. Future work can propose an automated environment for implementation of the steps mentioned and its application to case studies coming from the industrial world. The test management system modeled in this paper help to bridge the gap between the black box and white-box approaches, enabling execution behavior to be modeled for accuracy, component upgrades and updates. The model's easy adaptation to software evolution without a need of retesting the complete system saves time and

effort. In summary, the new test management system provides increased reusability of software test components and reduced time to market for applications.

ACKNOWLEDGMENTS

The author would like to thank the members of both the management and technical committees of various industry projects who worked together and contributed with numerous useful suggestions towards the successful completion of this research.

REFERENCES

[1] Boris Beizer, Software Testing Techniques, VanNostrand, second edition, 1990.
[2] Ball, T.,et al., "State Generation and Automated Class Testing. Software Testing", Verification and Reliability, Volume 10, Number 3, September 2000, 147-170.
[3] Biggerstaff, Ted J. "Software Reusability: Applications and Experience", ACM Press Frontier Series, 1989.
[4] John D. McGregor, "Verification and Validation Issues and Implications for Reuse", Technical Report, Department of Computer Sciences, Clemson University, Clemson, SC 29634.
[5] http://www.StickyMinds.com Original\Article info Reengineering Test Management.htm
[6] P. D. Johnson, "Mining legacy systems for business components: architecture for an integrated toolkit", Proceedings of the 26th Annual International Computer Software and Applications Conference (COMPSAC'02).
[7] M. R. Olsem, "An incremental approach to software systems re-engineering", Journal of Software Maintenance: Research and Practice, 10:181–202, 1998.
[8] Elizabeth Burd and Malcom Munro, "Investigating component-based maintenance and the effect of software evolution: a reengineering approach using data clustering", Proceedings of the International Conference on Software Maintenance, ICSM98, pp. 199-207. 1998.
[9] Santiago Comella-Dorda, Grace Lewis, Pat Place, Dan Plakosh and Robert Seacord,"Incremental Modernization for Legacy Systems", Technical Report CMU/SEI-2001-TN-006, Software Engineering Institute, Carnegie Mellon University, July 2001.
[10] Arnold, R. S, A Road Map Guide to Software Reengineering Technology," Software Reengineering, S. Arnold (ed.), IEEE Computer Society Press, 1993.
[11] Basili, V. R., Caldiera G., and Cantone, G. "A Reference Architecture for the Component Factory," ACM Transactions on Software Engineering and Methodology, vol. 1, no. 1, pp. 53-80.,1992.
[12] Beck, J. and Eichmann, D, "Program and Interface Slicing for Reverse Engineering", Proceedings of International Conference on Software Engineering, Baltimore, MD, May1993, pp. 509-518.
[13] Chikofsky, E. and Cross, J. H, "Reverse Engineering and Design Recovery: A Taxonomy,"IEEE Software, vol. 7, no. 1, 1990,pp. 13-17.
[14] Sneed, H. M., "Economics of Software Re-engineering," Journal of Software Maintenance: Research and Practice, vol.3, no.3, pp.163-182,1991.
[15] Wegner, P., "Capital-Intensive Software Technology", IEEE Software, vol. 1, no. 3, 1984.
[16] Weisskopf, M., Irving, C. W., McKay, C. W., Atkinson, C. and Eichmann, D.,"Maintenance In a Dual-Lifecycle Software Engineering Process,"Conf. on Software Maintenance, Monterey, CA,pp.4-8, November 1996.
[17] Patrick A.V. Hall and Lingzi Jin,"The Re-engineering and Reuse of Software", Software Engineering, Vol .1, p.355
[18] Prem Devanbu, "Analytical and Empirical Evaluation of Software Reuse Metrics" ,Proceeding of the 18th International Conference on Software Engineering (ICSE'96) p. 189,1996.
[19] Marcus A. Rothenberger, "Strategies for Software Reuse: A Principal Component Analysis of Reuse Practices", IEEE p.825, Transaction on Software Engineering, Vol. 29 No. 9, September 2003.
[20] F. Abbattista, G. M. G. Fatone, F. Lanubile, G.Visaggio, "Analyzing the application of a reverse engineering process to a real situation", Proceedings of the 3rd Workshop on Program Comprehension, Washington, D.C., November 1994, pp.62-71.
[21] V. R. Basili, "Viewing maintenance as reuse oriented software development", IEEE Software, January 1990, pp.19-25.
[22] E. J. Byrne, "A conceptual foundation for software re-engineering", Proceedings of the Conference on Software Maintenance, Orlando, Florida, November 1992, pp.226-235.
[23] M. M. Lehman, "Programs, life cycles, and laws of software evolution", Proceedings of the IEEE, Vol.68, no.9, September 1980, pp.1060-1076.
[24] Amir Tomer, Leah Goldin, Tsvi Kuflik, Esther Kimchi, and Stephen R. Schach, "Evaluating Software Reuse Alternatives: A Model and Its Application to an Industrial Case Study", IEEE transactions on software engineering, vol. 30, no. 9, September 2004,pp.601-612.

Jasmine K.S born in the Ernakulam District of Kerala state on October 14th in the year 1971. She received BSc degree in Mathematics from Mahatma Gandhi University, Kerala in 1991, MSc degree in computer science from Kerala University, Kerala in 1994 and M.Phil degree in computer science from Bharathidasan University, Tamilnadu in 2005. She is currently doing her PhD in computer science in Mother Teresa University, Kodaikanal, Tamilnadu.

She is an Assistant professor in the Department of MCA, R.V.College of Engineering, Bangalore. Since from 1995, she is working as a lecturer in the field of computer science. During 98-99, She held a visiting faculty position at Visveswarapura College of science, Bangalore. She has authored 23 research papers in the national and international level journals and conferences. Her research interests include Software reuse, Software performance, Software testing, data mining and experimental software engineering.

Ms. Jasmine is the member of Indian society for technical education (ISTE), Computer society of India (CSI), International Society for Computer Applications (ISCA), International Association of Engineers (IAENG) and International Association of Computer Science and Information Technology (IACSIT).

Dr.R.Vasantha received BSc degree, majored in Physics, Chemistry and Mathematics from University of Mysore, India in 1976, MSc degree in Mathematics from Manasa Gangotri, University of Mysore, India in 1978 and PhD from Indian Institute of Science, Bangalore, India in 1985.

She is a professor in the Department of Information science and Engineering, R.V.College of Engineering, Bangalore. She got more than 25 years of research experience. During Oct.1991-Oct.1994, she worked as a Scientist in National Aeronautical Laboratory, Bangalore, India, on Turbulence modeling of aerofoil.Sept1988-Sept.1991: Worked as Research Associate in the Dept. of Mechanical Engineering, University of New South Wales & University of Sydney, Sydney, Australia on turbulence modeling. Oct.1987-Sept.1988: Worked as a Senior Research Associate in the School of Mathematics, University of East Anglia, Norwich, England, on the initiation of detonation waves, Oct.1986-Oct.1987: Worked as a Senior Visiting Fellow in the School of Mathematics, University of East Anglia, Norwich, England. Aug.1985-Aug.1986: Worked as Research Associate in Dept. of Aerospace Engineering, IISc, Bangalore, India. She also has many years of teaching experience. During 1978-1980, she worked as a Lecturer, 1994-2002: Worked as Associate Professor of Mathematics, 2002-2006: Worked as Professor & HOD of Mathematics in an Engineering College in India. Her research interests are in the field of computational fluid mechanics. She is the author and coauthor of several publications appearing in international journals, books, and conference proceedings in the fields of Applied Mathematics and Computational fluid mechanics.

Dr. Vasantha is a Gold medal contender for PhD dissertation. And also she Won Scholarship from B.Sc. till the end of Ph.D.