Applying an Innovative Approach to Find the Reuse Rate of IT Enterprises

Pratyush Parida¹, Sidhartha Sankar Samal², P.Santhi Priya³
¹B.Tech, Electronics and Communication Engineering, GITAM University, Visakhapatnam.
²B.Tech, Electrical and Electronics Engineering, Kalam Institute of Technology, Berhampur.
³M.Tech, Software Engineering, GITAM University, Visakhapatnam.

Abstract - Reusability is the use of existing assets in some form within the software product development process. The ability to reuse relies in an essential way on the ability to build many things from various parts, and being able to identify the common things among those parts. Reusability is often an important feature for platform software. Software reuse is the process of creating software systems from existing software rather than building them again. Software reuse is still a developmental process. It appears in many different forms from ad-hoc reuse to systematic reuse, and from white-box reuse to black-box reuse. In this we do source code and design reuse with classes, application development, and design patterns. Software components provide a path for planned and systematic reuse. The term component is used as a synonym for object, module or function. Systematic software reuse and the reuse of components influence almost the whole software engineering process Software process models were developed to provide guidance in the creation of high-quality software systems by teams at predictable costs. With increasing reuse of software, new models for software engineering are practised. New models are based on systematic reuse of well-defined components that have been developed in various projects

Keywords— Modules, Functions, Cost, Quality, Source code, Reusability.

I. INTRODUCTION

Software is constantly increasing in its importance in product development. The study of Rine and Sonnemann showed the highest motivation for implementing software reuse were based on reduction of development and maintenance cost. Tomer and Etal proposed cost model for comparing three types of reuse mode: systematic, ad hoc and controlled.

The model was used to compare the potential costs and benefits of alternative reuse scenarios. However, there is no single empirical study verifies reuse will directly bring down development cost. Therefore, it needs further review and testing on the relationship of reuse to cost benefit.

Hence, the following hypotheses shall be further tested and reviewed with empirical data for the software development of the project:

• High Reuse rate will improve the competence
• High Reuse rate will improve the quality
• High Reuse rate will reduce the cost
• High Reuse rate will reduce the time to deliver projects

Based on these hypotheses we will have a clear idea about the software project development. Considerably, major software development is done through the IT Enterprises. Further, this Reuse concept will solve the areas like insufficient human resources, lack of development and supporting environment, lack of budget.

Benefits of Reuse further are classified as:

• Effective use of programmers:- Instead of programmer doing the same work on different projects, these programmers can develop reusable software that encapsulates their knowledge.
• Accelerated development:- Bringing a system to market as early as possible is often more important than overall development costs. Reusing software can speed up system production because both development and validation time should be reduced.

II. SYSTEMATIC VIEW

Software reuse, the use of existing software artifacts or knowledge to create new software, is a key method for significantly improving software quality and productivity. In this, we assume the actual design of the system and for implementation to the system many components are need to be developed. After developing the structure of the system we have to search for reusable components.

Some metrics are being used to calculate the reuse rate and incorporating the performance of the system in various scenarios.
FIGURE 1.1: Systematic view of reuse structure

III. METRICS USED IN DEVELOPING SYSTEM WITH REUSE

The major goal for reuse metrics and the emphasis on truthfulness and simplicity,

(i) Provide realistic measures of reuse: When we place a numeric value on our reuse accomplishments we want to have faith in that number. It should reflect our true reuse level without influence from other tangential factors.

(ii) Estimate the benefits of reuse: Nothing makes a believer out of people like seeing the benefits laid out in dollars. The metrics need to help us show the effect reuse has on the "bottom line." Exact values matter less than reasonable estimates.

(iii) Provide feedback to developers and management: Developers need a defined method to measure and report what they have done. People need feedback. This gives a sense of accomplishment and helps define their mission.

(iv) Give simple, easy to understand values: Metrics should not require a specialist from accounting to interpret. Getting people to adopt a new technique or idea often depends on how well they comprehend it. People will more readily adopt metrics that they understand and that provide meaningful values.

(v) Have a minimal impact on business: No one wants to make a project out of metric collection and reporting. Whatever data you need and metrics you generate must fit easily into the current business environment.

(vi) Comply with the tenants of metric theory: The metrics should yield consistent, repeatable values independent of who calculates them and should prove mathematically and/or axiomatically sound [Fenton91], [Weyuker88]. The details and importance of this statement probably do not interest the average practitioner. Among other things, we need it to assure us that our equations make sense and values they give don't go strange things like go down when we expect them to go up.

(vii) Encourage reuse: According to metric theory, metrics should not serve a prescriptive role. In other words, metrics should not tell people how to behave; they should objectively and accurately reflect attributes about the software. However, metrics affect people's behaviour and because of this we have found metrics a valuable tool when building a reuse program.

Software Reuse Metrics

Reuse rate is defined as the size of reused assets divided by the overall software size. Reuse Rate is the same and mostly used in reuse measurement. The reuse-process is among other things supported by metrics, that an assessment is viewed to the reusability of existing software-components. The so-called reusability-metrics become classified into empirical and qualitative methods, they are presently on the other hand module- or has component-oriented.

Reuse Rate (RR) = (Modules Reused without Revision)/(Total Number of Modules for a Project)

Productivity Metrics: There are two types of productivity measurement for software reuse.

(i) Gross productivity

(ii) Net productivity

Gross productivity: It measures the total program size including reuse part per effort required for developing the software.

Net productivity: It only measures the newly developed program size per effort required for such development.
Productivity (PD) = KNCSS / Person-day

Where, KNCSS = Thousand Non-commentary source statement

Person – day = Programming time of Each Developer

Quality Metrics: Identifying the Defects and Defects Density will show the quality of development. Defect Density defined as Number of Defect or error per non-commentary source statement. Defect means any part of the software that caused the software malfunction. If a defect in a component is detected during integration or system testing, or later during maintenance, Ericsson programmers write a Trouble Report that describes the defect in detail. The report typically contains information such as the severity of the defect, the assumed origin of the problem, the amount of time needed to correct the defect, and a defect code that assigns the problem to a trouble category, such as coding, wrong design rule, or documentation problem. The analysis shows that reused components had a lower defect density than non-reused ones, with defect density calculated by dividing the number of defects by the number of lines of code. At the same time, however, the reused components had more defect with high level of severity than the total distribution, but less of these defects after delivery.

Defect Density (DD) = No. of Defects / KNCSS

Cost Metrics: There are three parts of costs measurements here,

(i) First is the cost mining of the project developed by individual developers and cataloguing the assets in a repository.

(ii) Second is the cost of acquiring a copy of the project from the repository.

(iii) Third part will be the effort in reusing the project modules in white-box style.

Cost of direct effort for reuse = Effort in person-day × Salary rate of developer.

Where, Effort = (No. of Working days / Percentage of work completed) × 100.

Total Cost = Cost of management and support shared based on the duration of the project + Cost of direct effort for reuse and development of the software.

Project duration metrics: This metric is very straightforward and just directly measures the overall project duration in terms of calendar days and normalized with the KNCSS. It is defined as:

\[ \text{Duration} = \frac{\text{Duration in Calendar days} \times 100}{\text{KNCSS}} \]

IV. DEVELOPMENT PROCESS

Data is presenting with experimental proof for all the components developed for the software. Any further revision of software system has been reused with calculated values. The environment is quite stable and the reuse process is the same throughout the data evaluation.

Initial consideration of project

The data for 10 projects was collected with detail information of their reuse rate, productivity, quality, cost and duration. The reuse is in controlled mode the effort of the developers needed to include the search of the assets in repository, study and review of the assets, dissemination and retrieval of the right modules for modification or reuse. The gross productivity measurement is more appropriate for the current study as the effort measured is a total effort including all the training, review, programming time of each developer. Hence, gross productivity is defined as the software program size divided by total direct software development effort. Where software program size is defined as KNCSS (Thousand Non-commentary source statement) and direct software development effort is defined as person-day of software developer directly involved in the project.

<table>
<thead>
<tr>
<th>No. of projects</th>
<th>KNCSS</th>
<th>Total no. of modules</th>
<th>No. of Modules reused</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project-1</td>
<td>264</td>
<td>12</td>
<td>6</td>
</tr>
<tr>
<td>Project-2</td>
<td>538</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>Project-3</td>
<td>1056</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>Project-4</td>
<td>1022</td>
<td>13</td>
<td>6</td>
</tr>
<tr>
<td>Project-5</td>
<td>600</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>Project-6</td>
<td>1054</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>Project-7</td>
<td>434</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>Project-8</td>
<td>780</td>
<td>12</td>
<td>9</td>
</tr>
<tr>
<td>Project-9</td>
<td>654</td>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>Project-10</td>
<td>890</td>
<td>8</td>
<td>5</td>
</tr>
</tbody>
</table>
Reuse rate

The efficiency of Reuse rate metrics based reuse process refers primarily to the access statistics. Otherwise, the manual collection of reuse figures dominates, although the (statistical) analysis of these figures is generally also made with the help of a computer.

<table>
<thead>
<tr>
<th>Modules Reused</th>
<th>Total no. of modules</th>
<th>Percentage Reuse rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>12</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>25</td>
</tr>
<tr>
<td>5</td>
<td>15</td>
<td>33.3</td>
</tr>
<tr>
<td>6</td>
<td>13</td>
<td>46</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>40</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>18.6</td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td>23</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>38.5</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>22</td>
</tr>
</tbody>
</table>

Productivity Table

Every developer in the company needs to fill in the timesheet that records the detail of their work with accuracy up to an hour. The database consists of the following information: project name, nature of work and Time period. Nature of work is subdivided into five categories: source code review, development, debugging, training and documentation.

Cost table

All the costs were reviewed in raw values but presented as normalized figures so the final reported data is in normalized index form. Cost index is calculated by acquiring the direct salary of different developer that reflects their experience in the field. The indirect cost accounts for the shared management and support cost. After reviewing the actual finance data of the team, the management and support team salary and expenditure is shared with all projects handled by the team according to project duration.
The reuse is managed in project level but not with a fully defined reuse system and metric collection for supporting the work, it is hard to classify this as either systematic or ad-hoc reuse. Tomer et al. defined the controlled reuse process. A core-asset repository shall be established in which assets are stored and cataloged for the benefit of other products. There is an individual party mining all private assets and cataloging them in the repository. The developer will get a copy of the asset in original form and reuse it in white-box style. White box reuse means reuse of components by modification and adaptation while black-box reuse means reuse in its functionality. Ericsson-Norway collects a large amount of data relating to its software. Much of this data is awaiting analysis, which represents a mine of information on product characteristics and quality.

An analysis by Parastoo Mohagheghi sifted through Ericsson’s data to try to answer the question: what effect does software reuse have on product defects and stability? Answer, is that software reuse significantly improves quality. Companies like Ericsson are increasingly moving toward component-based software engineering (CBSE), where related products and systems can be assembled from pre-built components. These reusable components can take a variety of forms, from existing software libraries, to free-standing commercial, off-the-shelf products (COTS) or open-source software (OSS), to entire software architectures and their components. CBSE promises many advantages, such as a shortened product development time, reductions in total costs and since new software components can be purchased instead of developed in fast access to new technology. Mohagheghi’s examine one release that comprised 470,000 lines of non-commented Code (KNCs), of which 64 percent was in Erlang, 26 percent was in C, and the remainder was in other programming languages like Java or Perl. Other researchers have assumed that reused components would change more often than non-reused components because the reused components had to meet the requirements of several different systems. However, Mohagheghi’s analysis also showed that reused components were far less likely to be modified between successive releases, as measured by the percentage of code that was modified. This meant the reused components were far more stable than their non-reused counterparts.

V. REVIEW AND STUDY

The reuse is managed in project level but not with a fully defined reuse system and metric collection for supporting the work, it is hard to classify this as either systematic or ad-hoc reuse. Tomer et al. defined the controlled reuse process. A core-asset repository shall be established in which assets are stored and cataloged for the benefit of other products. There is an individual party mining all private assets and cataloging them in the repository. The developer will get a copy of the asset in original form and reuse it in white-box style. White box reuse means reuse of components by modification and adaptation while black-box reuse means reuse in its functionality. Ericsson-Norway collects a large amount of data relating to its software. Much of this data is awaiting analysis, and represents a mine of information on product characteristics and quality.

VI. CONCLUSION

Software Reuse evaluates the developmental view of the concept reuse rate that data to be collected in less time complexity from central repository which is treated as data centre and the data collected in required and relevant chunks. This study verifies Quality, Productivity and Cost all having strong linear relationship with module reuse. However, there is no significant relationship between software reuse rate and development duration.
REFERENCES

[1] https://www.google.co.in/search?q=Rine++and++Sonnemann&ie=utf-8&oe=utf-8&rls=org.mozilla:enUS:official&client=firefox-a&channel=fflb&gfe_rd=cr&ei=Yf9YVOOgI86GvAS2mYGAg

[2] https://www.google.co.in/?gfe_rd=cr&ei=GQJZVOEPNTCuAS9f4GwCg&wsv_rd=ssl&q=tomer%20et%20al%20software%20metrics


[10] Hironori Washizaki1, Hirokazu Yamamoto2 and Yoshiaki Fukazawa1Department of Computer Science, Waseda University3-4-1 Okubo, Shinjuku-ku, Tokyo 169-8555, Japan washi, fukazawa@fuka.info.waseda.ac.jp, 2Matsushita Electric Industrial Co., Ltd.1006 Kadoma, Kadoma City, Osaka 571-8501, Japan

[11] IR&D Department, Computime Ltd., Hong Kong 2Department of Systems Engineering and Engineering Management, City University of Hong Kong, Hong Kong (wha@computime.com, mehsun@cityu.edu.hk, minxie@cityu.edu.hk)


[13] Shlomit Moradi and Tsvi Kuflik1Orbotech, 2University of Haifa 1shlomit@orbotech.com, 2kuflik@is.haifa.ac.il
