SUMMARY

DESIGN OF MODEL(S) RE-ENGINEERING OBJECT-ORIENTED SOFTWARE SYSTEMS

Re-engineering is young and very prospective area in the coming future. There are number of potential benefits from re-engineering software systems. Software engineers and developers are engaged in developing new software systems. The technology is changing fast and with this change, business processes are changing, our requirements are changing, work platforms are changing. In the fast changing world, today’s software can be old tomorrow and new software is required. This is huge financial investment burden on the client organizations. Re-engineering is the need of the hour in this fast changing scenario.

If software is reengineered at the right time, re-engineering cost is feasible (minimum cost) and the system is reconstituted to a new form (new one) to work for another life span. As software system does not wear or tear out, it can be reengineered many times (when it gets old). If this happens, the software investments graph of client organizations can be lowered significantly by re-engineering practice.

Software re-engineering models exist, but they were less applicable because the software development technology is changing fast. Existing models were mostly for procedural oriented software systems and many software industries have shifted to object-oriented paradigm. Therefore it was required to design new model(s) for re-engineering of objects-oriented software systems.

1 Objectives of the study

1.1 Main Objectives

The main objectives of the study are to reduce software investments in the client organizations by re-engineering the legacy software systems and to escape new software developments. Maintenance cost of the software can also be decreased significantly. Software age can be increased many folds by re-engineering as and when it gets old. In this way, software system once purchased can be used for long time.
1.2 Sub Objectives

Model(s) are designed for various stages and activities in the re-engineering process. Model(s) will assist software engineers and software managers in re-engineering object-oriented legacy software systems. This area has enormous importance as software is increasing every where and with fast changes re-engineering is needed as software can not be thrown away with small changes.

Re-engineering can be made well-liked, easy, and less expensive by developing the precise model(s). Highly sophisticated software tools, techniques, and methods must be developed to promote re-engineering.

In general, following are the objectives of re-engineering object oriented software systems.

- Functional enhancement
- Improve maintainability
- Improve customer satisfaction
- Migration to other platform
- Improve reliability
- Reduce the software maintenance cost
- To escape costly new software development

2 Proposed Model(s)

New model(s) for re-engineering object-oriented software systems have been proposed in this research work. It was pre-requisite to study the existing models for re-engineering software systems. From time to time software re-engineering model(s) were proposed by software engineers and researchers. As software development technology changed, existing models were less applicable in the present scenario. Existing models were applicable for re-engineering software systems in procedure oriented languages and the software industry is shifting to object-oriented paradigm. The study found many limitations of the existing re-engineering model(s). The existing models do not address the following issues.
• When to re-engineering the software system?
• What will the estimated cost of re-engineering?
• The efforts of re-engineering in comparison to new development of equal domain
• There is no uniformity and interoperation among the existing models, it work in stand alone mode.
• Existing models were for re-engineering of the whole software system
• There were many gaps in re-engineering process activities.
• Effects of re-engineering on various parameters of software were ignored in existing models.

On the basis of study of existing re-engineering models, a set of model(s) is proposed for complete re-engineering process. Case studies on re-engineered software projects were conducted to evaluate the proposed models.

Following are the proposed models:

1) Model for Differentiating ‘Maintenance’ and ‘Re-engineering’

From literature review, it was seen that there was confusion over re-engineering and maintenance of software systems. Software managers of small and medium sized client companies feel little difference in maintenance and re-engineering and even some time use these two terms interchangeability. Re-engineering is very potential and promising area and is gaining importance. In this work these issues were addressed and it will be very useful for vendors (software companies) and the clients (software managers). Three model(s) have been proposed which differentiate the two terms. Following are the three models for differentiating the terms ‘Maintenance’ and ‘Re-engineering’.

• Model based on Software age
• Model based on Software Life Cycle (SLC)
• Discrete model for Differentiation
With the help of this model software managers will take ‘maintenance’ and ‘re-engineering’ terms differently. They will be aware about re-engineering.

2) Decision Making Model(S) for Re-engineering

Three model(s) are proposed for making re-engineering decisions at the right time. Right time re-engineering cost is optimal (minimum) cost. Following are the three decision making models

• Thoroughfare decisive point
• Decisive Point based on Maintenance cost
• Decisive Point based on faulty objects

Software managers can know the right time for re-engineering with the help of this model and therefore software system will be re-engineered at the right time.

3) Re-engineering Efforts Comparison Model

This model compares re-engineering efforts of candidate software and new development of software of equal domain. It is useful to software managers to choose the right alternative. On comparing efforts for re-engineering and for new development of equal domain, benefits of re-engineering can be seen in terms of technical resources.

Re-engineering will be attractive as one forth efforts are required for re-engineering according to this model.

4) Cost Estimation Model

Cost estimation model estimates the re-engineering cost of the candidate software in person-month. Cost estimation is pre-requisite for initiating re-engineering project.

It will help the software managers and software engineers in negotiating re-engineering projects.

5) Rainbow Model for Re-Engineering

This is economical model for re-engineering legacy object-oriented software systems. In this model, instead of re-engineering the whole software, only faulty
objects are re-engineered. Faulty objects are re-engineered individually and independently.

Software engineers can use rainbow model for reengineering object-oriented software system.

6) Model for Increasing Software Age

Client organizations can increase software age manifolds and can reduce software investments heavily. Re-engineering of software system can be done when it gets old (aged).

Once purchased software can be used for long time according to this model.

Proposed models for various re-engineering stages/phases successively show complete and smooth path for re-engineering object-oriented software systems. This work will be useful for software development companies as well as for client companies.

3 Case Studies

Case studies were conducted to estimate the effects of re-engineering on old legacy software systems. Various aspects of legacy software systems are compared with the aspects after re-engineering. Effects of re-engineering can be seen on software investments and many other parameters of software systems. The results are interesting and fruitful on various aspects of re-engineered software systems. More accuracy and financial benefits can be achieved by re-engineering. Results are based on empirical data collected from small and medium sized software companies and client companies in the region of Punjab, Chandigarh and Haryana. There are many parameters related to software systems that can be estimated in comparison to legacy software systems. It shows the improvement in the re-engineered software system which is almost equivalent to new software system. The following parameters of re-engineered software systems were estimated.

- Performance of software
- Security of software
- Interface ease
• Maintenance cost
• Manual work decrease
• Complexity of software
• Investment on software

Re-engineering Status

Case studies were also conducted to know the Software re-engineering status in comparison to new software development.

Interest of software engineers and developers

Interest of software developers and engineers is more in software development than in software re-engineering.

Problems in re-engineering

In this work, 32 statements formulated relating to reengineering problems. As re-engineering practice is low as compared to software development from scratch, it was desired to know the reasons for low status. Case study conducted to find the most effective and stable statements among the 32 statements (given in chapter VI section 6.3). Eight most effective and stable statements were arranged according to their severity and stability.

4 RESULTS AND DISCUSSION

4.1 Facts Established in the Proposed Model (s)

• Re-engineering efforts for object-oriented software systems are significantly less than the efforts of developing software from scratch.
• Life of software can be increased many folds by re-engineering.
• Right time re-engineering cost is minimum cost.
• Re-engineered software is equivalent to new software and works for another life span.
• Re-engineering can be achieved by re-engineering defective part only.

4.2 Case Study Results
4.2.1 Increased Parameters

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Software Aspect</th>
<th>Increase</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Performance of software</td>
<td>63.5 %</td>
</tr>
<tr>
<td>2.</td>
<td>Security of software</td>
<td>55.62 %</td>
</tr>
<tr>
<td>3.</td>
<td>Interface ease</td>
<td>56.2 %</td>
</tr>
</tbody>
</table>

Table 1

4.2.2 Decreased Parameters

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Software Aspect</th>
<th>Decrease</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Maintenance cost</td>
<td>51 %</td>
</tr>
<tr>
<td>2.</td>
<td>Manual work</td>
<td>26.72 %</td>
</tr>
<tr>
<td>3.</td>
<td>Complexity</td>
<td>32.72 %</td>
</tr>
<tr>
<td>4.</td>
<td>Investment on software</td>
<td>69.1 %</td>
</tr>
</tbody>
</table>

Table 2

From the above results it shows that re-engineered software is constituted as new one as the software parameters are improved significantly.

4.2.3 Re-engineering status

<table>
<thead>
<tr>
<th>Small and Medium sized Software Companies</th>
</tr>
</thead>
<tbody>
<tr>
<td>New development work during last 5 years</td>
</tr>
<tr>
<td>Re-engineering work during last 5 years</td>
</tr>
</tbody>
</table>

Table 3

Re-engineering status is low as compared to software development
4.2.4 Interest in software development / re-engineering

<table>
<thead>
<tr>
<th>Software engineers/developers</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(Small and medium sized software companies)</td>
<td></td>
</tr>
<tr>
<td>Interest in new development</td>
<td>61.2</td>
</tr>
<tr>
<td>Interest in Re-engineering</td>
<td>38.8</td>
</tr>
</tbody>
</table>

Table 4

Software developers and engineers show more interest in software development than re-engineering of legacy software.

4.2.5 Factors which hinder re-engineering

The most effective and stable statements (reasons) were found out among the 32 statements. The statements are eight in number and are given in the following table according to the severity. If these notable statements are resolved, re-engineering will be promoted significantly.

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Statement No.</th>
<th>Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Statement-2</td>
<td>Re-engineering is difficult without source code</td>
</tr>
<tr>
<td>2</td>
<td>Statement-24</td>
<td>Risks of negative results if re-engineering fails</td>
</tr>
<tr>
<td>3</td>
<td>Statement-23</td>
<td>Legal problem in copying the source code if re-engineering is done by some other company</td>
</tr>
<tr>
<td>4</td>
<td>Statement-1</td>
<td>Re-engineering practice is not common due to non availability of source code</td>
</tr>
<tr>
<td>5</td>
<td>Statement-20</td>
<td>Re-engineering takes time</td>
</tr>
<tr>
<td>6</td>
<td>Statement-13</td>
<td>Software reengineering is often associated with business process reengineering (BPR)</td>
</tr>
<tr>
<td>7</td>
<td>Statement-31</td>
<td>Paying less attention to customers demands and depending upon legacy software</td>
</tr>
<tr>
<td>8</td>
<td>Statement-11</td>
<td>Estimated reengineering cost is not known</td>
</tr>
</tbody>
</table>

Table 5
4.2.6 Other findings in Case Studies

- Companies solely for re-engineering are not seen in the region under consideration.
- ‘Re-engineering’ and ‘maintenance’ terms are used interchangeably by some software managers and users.
- Lack of automated re-engineering tools
- Potentials of re-engineering are out of sight
- There is legal problems in copying source code
- Software companies hesitate to give information on cost related issues.

These findings provide valuable information to software engineers, managers and practitioners for re-engineering software systems. This type of information is also very useful for researchers and academicians for further research in the field of software engineering and software re-engineering. Empirical data was collected during the efforts to directly address the problems in re-engineering.

Empirical study was lacking in the field of software re-engineering which will be very useful for researchers and academicians for further studies.

5 Shortfalls/Gaps in the Existing Models in the light of Proposed Model(s)

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Issues</th>
<th>Proposed model(s)</th>
<th>Existing models</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Re-engineering Vs. Maintenance</td>
<td>Differentiated the two terms on various issues on software</td>
<td>No such model is existing</td>
</tr>
<tr>
<td>2.</td>
<td>Re-engineering cost</td>
<td>Cost is based on re-engineering faulty objects only</td>
<td>Cost is based on re-engineering the whole software</td>
</tr>
<tr>
<td>3.</td>
<td>Re-engineering efforts</td>
<td>Re-engineering efforts are one forth of the efforts for new development from scratch</td>
<td>No such model is existing</td>
</tr>
<tr>
<td>4.</td>
<td>Software Age</td>
<td>Software age can be increased many folds</td>
<td>Software age related issue is discussed.</td>
</tr>
<tr>
<td>5.</td>
<td>Re-engineering approach</td>
<td>Trickle down approach is mostly followed.</td>
<td>Big bang or Trickle down approach is followed</td>
</tr>
<tr>
<td>6.</td>
<td>Re-engineering Methodology</td>
<td>Re-engineering is based on re-engineering individual objects</td>
<td>Re-engineering based on whole software system</td>
</tr>
<tr>
<td>7.</td>
<td>Applications</td>
<td>Applicable to object-oriented software systems</td>
<td>Applicable to procedural oriented systems</td>
</tr>
</tbody>
</table>

Table 6
6 Suggestions

Modules, objects and frameworks should be more refined and cohesive so that re-engineering can be done individually and independently. Scope for re-engineering should also be kept in mind while developing software system.

New software cycle is defined after development stage called software life cycle (SLC). It is a period from software delivery to retirement of software. Software life cycle is when software is in working state. Software development life cycle (SDLC) is when software is in developing stage. It is suggested that maintenance stage which is last stage in Software Development Life Cycle (SDLC) must be extracted from SDLC and put in software life cycle (SLC). In this way, SLC is a cycle consisting of only two stages. Maintenance will be first stage and re-engineering will be the second stage in software life cycle (SLC).

More software re-engineering tools must be developed to reduce the re-engineering time, cost and risks. Although full automation is not possible, it can be achieved up to significant level.

From the case study, it was found that re-engineering practice is low as compared to new software development in the small and medium sized software companies. To promote this practice at the all levels, small, medium and big software companies must be made aware of the potential of the software re-engineering. In future, software will exist everywhere and if this practice is not encouraged, there will be crisis in software industry. Re-engineering should be encouraged in software industries and client industries as re-engineering is a linkage in software companies and client companies. Software industries, client industries and institutes come at a common platform for innovations in the field of re-engineering.

The number of software development companies is growing and there is no single software re-engineering company. Companies and institutes must be established solely for re-engineering.

The effective and stable statements (reasons) found for low status of re-engineering must be resolved to promote re-engineering.
The most effective and stable statement (Statement-2) is the first statement in the above table 6. It says ‘Re-engineering is difficult without source code’. If copy right and source code is provided with the software to the client industry at the time of software delivery, then re-engineering work will increase.

In this way if all the majors reasons found for low status of re-engineering are solved then re-engineering work will be increased up to large extent.

It is suggested that to encourage empirical research on re-engineering, software companies must cooperate to provide right information required by the researchers as this type of information is used for innovations and academic purpose only.

7 Future Work

The proposed re-engineering models depend upon manual work up to some extent. Once the models were designed, tools can be built around it to achieve automation in re-engineering up to some extent.

The values of the constants embedded in the proposed model(s) are to be determined which is a future work to be done. The applications of proposed model(s) can be measured and matched in the real situations. Metrics and measures can be added to this research work to measure the utilities of the proposed models.

An empirical study of the applications of the proposed models will help in improving the proposed models.

The model(s) proposed in chapter five are based on objects and are for re-engineering of object-oriented software systems. This work could be extended to support the applications of more complex design patterns and more refactoring strategies to re-engineering. It can be extended further to component based re-engineering software systems.

Primary data collected on various software aspects and re-engineering issues can be used for further research in the field of re-engineering of software systems.