Identification of Quality Improvement Strategies using COPQ in Software Industry

YEONG-SEOK SEO\textsuperscript{1*}, DONGHWAN SHIN\textsuperscript{1}, GOOKHYUN KIM\textsuperscript{2}, JONGMOON BAIK\textsuperscript{1}, DOO-HWAN BAE\textsuperscript{1}

\textsuperscript{1}KAIST, Daejeon, Republic of Korea
\textsuperscript{2}Financial Supervisory Service, Seoul, Republic of Korea

(Received on February 29, 2012 and revised on June 20, 2012)

Abstract: Software COPQ is one of the most important performance measures in Six Sigma or any key improvement effort. However, in practice, many software development organizations are having difficulty in estimating their valid software COPQ. Thus, in this paper, we investigate the methods, CONstructive QUALity MOdel (COQUALMO) and Defect Amplification Model (DAM) that are commonly used as the quality estimation model, to estimate software COPQ in a software development organization that starts establishing software quality improvement strategies. In addition, we derive software COPQ from the software defect data that collected from the organization and make an analysis of the additional experimental results by using the data. This study is currently being used for making decisions to set software quality improvement strategies in the organization.

Keywords: Software COPQ, COQUALMO, DAM, software process improvement.

1. Introduction

The development of high-quality software is an issue of growing importance throughout the software engineering communities. In particular, with the diversity of software development environment and the increase of software complexity, the importance of software quality management has been more emphasized than ever.

In order to develop high-quality software, there is a need to consider the software development and maintenance cost that will incur for a software development organization. As described in [1], there are two categories of Cost of Quality (COQ): the Cost of Good Quality (COGQ) and the Cost of Poor Quality (COPQ). COGQ is composed of the prevention cost and the appraisal cost:

- Prevention cost: The cost of activities to prevent software poor quality (e.g., quality planning, process capability evaluations, and staff training),
- Appraisal cost: The cost of activities to assure conformance to quality standards and performance requirements (e.g., design reviews, inspections, and product audits).

On the other hand, COPQ is composed of the internal failure cost and the external failure cost:

- Internal failure cost: The cost of activities to cope with errors discovered before releasing to customers (e.g., bug detection, bug fix, and rework),
- External failure cost: The cost of activities to cope with errors discovered after releasing to customers (e.g., product recalls, warranty claims, and processing customer complaints).

Based on the characteristics of COQ, Stephen T. Knox investigates the relationship...
between software quality cost and Capability Maturity Model (CMM) [2]. As presented in Figure 1, this ‘Knox model’ shows the above costs as a function of the CMM level. The x-axis and the y-axis indicate CMM levels and cost as a percentage of the software development budget, respectively. Note that, as the CMM level is lower, the COGQ rate is not changed largely while the COPQ rate is higher (The COPQ rate is 55% for CMM level 1, 45% for CMM level 2, 35% for CMM level 3, 20% for CMM level 4, and 6% for CMM level 5). That is, in most general software development organizations that do not acquire high CMM levels, high COPQ have an effect on overall software lifecycle cost. Thus, first of all, it appears that a focus on COPQ is very important and desirable to improve software quality. Consequently, the correct estimation of COPQ can contribute greatly to improve the business performance of the software development organization, by identifying and minimizing the activities that cause high COPQ.

**Figure 1:** The Knox Model of the Cost of Software Quality [2]

In this paper, we try to estimate software COPQ in a software development organization that starts establishing software quality improvement strategies, using CONstructive QUALity Model (COQUALMO) [5] and Defect Amplification Model (DAM) [6]. Moreover, we derive software COPQ from the software defect data that collected from the software development organization (called as the E company). Finally, we discuss the variation of software COPQ according to the difference of defect removal distribution and the effect of software process improvement.

The remainder of this paper is organized as follows: Section 2 explains the characteristics of COQUALMO and DAM as background to our work. Section 3

---

3 Generally, in manufacturing sectors, many organizations manage and control COQ from their quality improvement strategies, including COQ estimation models [3, 4].

4 Because of the limitation of collecting the data for the external failure cost, in this paper, software COPQ means the internal failure cost based on software defect data, except the external failure cost.
describes the derived results by applying COQUALMO and DAM. Section 4 discusses the additional experimental results. Section 5 presents threats to validity. Finally Section 6 concludes this paper and suggests future work.

2. Background

2.1 COQUALMO

COQUALMO (COnstructive QUALity MOdel) is one of famous software quality model, which is developed by Sunita Chulani [5] at Center for Software Engineering, USC (University of Southern California). This model requires the same drivers with COCOMO II (COnstructive COst MOdel II) to estimate software quality except for FLEX driver. COCOMO II produces the estimated development effort, cost, and schedule. On the other hands, the result of COQUALMO presents the estimated numbers of residual defects and defect density per unit of size. Figure 2 shows the schematic representation of COCOMO II and COQUALMO.

![Figure 2: The Concept of COCOMO II and COQUALMO](image)

We can calculate COGQ, COPQ and its ROI by applying COCOMO II and COQUALMO. COCOMO II can estimate the total development cost and their distribution for each phase and each activity. By extracting the costs of COGQ activities from the distribution, we can obtain the COGQ. On the other hands, if the cost to fix a defect is known for each development phase, COPQ can be obtained by multiplying it with the number of residual defects for each phase. ROI can be calculated when we obtain the ratio of reduced COPQ per COGQ for each RELY level, which is one of COCOMO II drivers. RELY indicates the required reliability of the target software system. This driver is appropriate to indicate the level of defect removal activities because the required reliability determines how much defect removal activity should be performed. The values of COGQ, COPQ, and ROI can be used to make a decision related to software quality improvement process.

2.2 Defect Amplification Model

Defect Amplification Model (DAM), developed by IBM in 1981, is a useful model to show that how defects are transferred and amplified in a subsequent software development life cycle (SDLC) [6]. The basic idea of this model is that software defects can be
introduced or removed in every phase of SDLC and the remaining defects from previous development phase are not only just transferred to next phase but also amplified. Thus, in every phase of a SDLC, we have two constant values: defect removal rate and amplification rate. Figure 3 shows the schematic representation of DAM.

![Figure 3: Defect Amplification Model](image)

As shown in Figure 3, there are three different types of defects in each software development phase: (1) Defects passed through, (2) Amplified defects, and (3) Newly generated defects. At first, defects from previous phase are divided into (1) and (2). In this paper, in order to determine how many defects are remained as (1), we use a defect residual rate (1-Defect removal rate). That is, if a defect residual rate is 30%, (1) is calculated as follows: (1) = Defects from previous phase * 30%. In addition, (2) is amplified by using an amplification rate that is generally determined by domain experts or historical project data sets. The amplification rate is referred to NASA’s work [7]. Finally, in order to determine (3), the results from COQUALMO are applied because there were no available data for (3) from the E company.

As a result, the total number of defects for this phase is the sum of (1), (2), and (3). We can finally calculate the number of defects passed through to next phase by multiplying the defect detection rate in the phase. Because of this property, defect detection ratio is also called a defect filtering effectiveness how many defects are detected in a phase.

3. Application of Models for COPQ Estimation

3.1 Results by COQUALMO

We applied COQUALMO to target software project for the E company to analyze COPQ and its ROI. However, in the E company, the information for the cost to fix a defect is not enough. Thus, we discuss this problem with a project manager of the E company and calculate the cost using the average salary of injected people.

An on-site interview was conducted with managers of the target development team to determine the drivers and defect removal profiles. The number of introduced, removed, and residual defects for each phase is described in Table 1.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Design</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Number of Defects Introduction</td>
<td>14.62</td>
<td>91.00</td>
</tr>
<tr>
<td>The Number of Defects Removed</td>
<td>10.62</td>
<td>69.39</td>
</tr>
<tr>
<td>The Number of Defects Remained</td>
<td>4.00</td>
<td>21.73</td>
</tr>
</tbody>
</table>
As shown in Table 1, introduced defects are removed appropriately in requirement and design phases. However, defect removal ratio is the smallest in the design phase. This means that we need more defect removal activities during design phase to improve software quality.

Figure 4 presents the result of the application of COCOMO II and COQUALMO. COGQ is increased when RELY is raised from low to very high. On the other hands, COPQ is decreased because more defects are removed in the early phases. The difference of slope for COGQ and COGQ is very important. This indicates that we can decrease COPQ effectively when we invest and manage software quality improvement process focused on COGQ. That is, if we increase COGQ, COPQ can be decreased significantly.

ROI is the highest when RELY driver is low even if COPQ is decreased as COGQ is increased. Moreover, ROI is decreased by stages from low to high, and it is increased when RELY is very high. This means that we should set up RELY as low if available resources are limited and the highest effect is required. On the other hand, we need to determine RELY level as very high, if the software requires very high reliability.

3.2 Results by DAM

Before we apply DAM using the data collected from the E company, we clarify some restrictions for the data.
- We have no information about newly generated defects for every SDLC phase.
- There is no classification with defect detection and removal, so we just use the number of defects removed for each development phase.
- Overall, SLOC is too small to expect reliable analysis result for COQUALMO.

Even though these restrictions exist, we can find interesting results from the DAM analysis. The defect data from the E company and the additional data for applying DAM are summarized in Table 2.
Table 2: Software Defect Data for DAM

<table>
<thead>
<tr>
<th></th>
<th>Requirement</th>
<th>Design</th>
<th>Implementation</th>
<th>System testing</th>
<th>Qualification testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defect residual rate</td>
<td>0.0%</td>
<td>86.8%</td>
<td>93.9%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Defect removal rate</td>
<td>100%</td>
<td>13.2%</td>
<td>6.1%</td>
<td>80%</td>
<td>80%</td>
</tr>
<tr>
<td>Newly generated defects</td>
<td>14.62</td>
<td>87.00</td>
<td>27.97</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Amplification rate</td>
<td>0.00</td>
<td>2.25</td>
<td>3.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

As shown in the first and the second row of Table 2, we can easily see that this company mainly focuses on two software testing phases to detect defects which can cause higher defect correction cost.

The estimation results by DAM are described in Figure 5. The rightmost value 4.23 is the number of remaining defects after the qualification testing. In order to compare with the average defect density of U.S. industry and Korea industry, we transformed SLOC into Function Point (FP) and then calculate the defect density from this final defect value. Table 3 shows defect densities of USA [8], Korea [9], and the E company.

Table 3: The Comparison of Defect Density (Unit: defects/FP)

<table>
<thead>
<tr>
<th></th>
<th>U.S. Industry Average</th>
<th>Korean Industry Average</th>
<th>The E company</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defect density</td>
<td>0.05</td>
<td>0.17</td>
<td>0.39</td>
</tr>
</tbody>
</table>

As summarized in Table 3, the defect density from the E company is 2.3 times more than Korea industry average and 7.8 times more than U.S. industry average. Note that this value means the software quality to be delivered to customers. As presented in Figure 5, the main reason why this company has such high defect density at the end is that defect removal effectiveness for design and implementation phases is too low so that lots of defects passed through to the end.

4. Discussion

4.1. The Effect of Defect Removal Distribution on COPQ

In a software development process, the cost to remove a defect at the last phase is much higher than that at the early phase. Thus, we investigate the effect of the distribution of defect removal activities in a SDLC by comparing two different product models. These two product models derive from the similar products of the E company. However, one is conducted with defect removal activities from the early phase of development and the other is not.

At first, we grasp defect removal cost in each software development phase. Table 4 shows the defect removal cost for analyzing our case study (The data is collected from a
company that is in the same field as the E company). Note that the cost described in Table 4 includes all the cost for documentation, confirmation, and others related reworks for removing a defect.

**Table 4: Defect Removal Cost (Unit: $1)**

<table>
<thead>
<tr>
<th>SDLC</th>
<th>Requirement</th>
<th>Design</th>
<th>Implementation</th>
<th>Unit Testing</th>
<th>System Testing</th>
<th>Qualification Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defect Removal Cost</td>
<td>150</td>
<td>150</td>
<td>150</td>
<td>5,100</td>
<td>9,000</td>
<td>24,000</td>
</tr>
</tbody>
</table>

As described above, we compare two product models with different defect distribution to find the effect on COPQ. The defect data with the SDLC is shown in Table 5. Note that Model A has no defect removal activities from requirement to unit testing phase.

**Table 5: The Number of Defects Removed with SDLC**

<table>
<thead>
<tr>
<th>Model</th>
<th>Requirement</th>
<th>Design</th>
<th>Implementation</th>
<th>Unit testing</th>
<th>System testing</th>
<th>Qualification testing</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>117</td>
<td>43</td>
<td>160</td>
</tr>
<tr>
<td>B</td>
<td>22</td>
<td>12</td>
<td>3</td>
<td>11</td>
<td>39</td>
<td>9</td>
<td>96</td>
</tr>
</tbody>
</table>

Based on the defect removal cost with the SDLC and the number of defects for two product models, we calculate COPQ by multiplying the defect removal cost and the number of defects. For example, to calculate COPQ in System testing of Model A, the defect removal cost in System testing and the number of defects in System testing are multiplied. In addition, the generated value is normalized by the total number of defects. That is, 9,000 * 117 / 160 = 6,581.25. Table 6 shows the estimated COPQ for each phase.

**Table 6: The Number of Defect based on COPQ for Two Product Models (Unit: $1,000)**

<table>
<thead>
<tr>
<th>Model</th>
<th>Requirement</th>
<th>Design</th>
<th>Implementation</th>
<th>Unit testing</th>
<th>System testing</th>
<th>Qualification testing</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>6.58</td>
<td>6.45</td>
<td>13.03</td>
</tr>
<tr>
<td>B</td>
<td>0.03</td>
<td>0.02</td>
<td>0.005</td>
<td>0.58</td>
<td>3.66</td>
<td>2.25</td>
<td>6.55</td>
</tr>
</tbody>
</table>

As noted in Table 4, the defect removal cost in qualification testing is 160 times higher than that in requirement, and this implies the striking result of total COPQ. As shown in Table 6, Model B reduces about 50% of COPQ of Model A by defect removal activities in early phase of software development. Because these two product models are similar, we successfully infer that if the E company conducts defect removal activities like the process used in Model B, they can reduce about 50% of COPQ. Consequently, when we compare two product models which have different defect removal distribution, Model B is better than Model A in terms of COPQ.

4.2 The Effect of Software Process Improvement on COPQ

In order to extend the implication of the previous section, we also focus on the effect of software process improvement on COPQ, based on the Model A. By assuming that software process improvement makes more defect removal activities at relatively early phase of software development, we simply increase the number of removed defects to 5% (and 10%) for each requirement, design, implementation and unit testing, and decrease the number of removed defects to 10% (and 20%) for each system and qualification testing. Table 7 shows the data simulated by software development. We call the 5% increasing...
version as “5%-shift” and 10% increasing version as “10%-shift”.

<table>
<thead>
<tr>
<th>Name</th>
<th>Requirement</th>
<th>Design</th>
<th>Implementation</th>
<th>Unit Testing</th>
<th>System Testing</th>
<th>Qualification Testing</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base (A)</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>73.1</td>
<td>26.9</td>
<td>100</td>
</tr>
<tr>
<td>5%-shift</td>
<td>5.0</td>
<td>5.0</td>
<td>5.0</td>
<td>5.0</td>
<td>63.1</td>
<td>16.9</td>
<td>100</td>
</tr>
<tr>
<td>10%-shift</td>
<td>10.0</td>
<td>10.0</td>
<td>10.0</td>
<td>10.0</td>
<td>53.1</td>
<td>6.9</td>
<td>100</td>
</tr>
</tbody>
</table>

Using the simulated data described in Table 7, we calculate COPQ in the same way as described in Section 4.1. The result is summarized in Figure 6.

![Figure 6: Prediction of COPQ with Software Process Improvement](image)

Note that although we redistribute defect removal activities on the early 4 phases, there is no significant increase on COPQ for these phases. However, the cost to fix for the system and qualification testing is substantially reduced. That is, it is necessary to focus on the defect removal activities in the early phases of a SDLC, to take the opportunity for significant cost reduction in software development projects.

5. Threats to Validity

There are several threats to the validity of this work. First, in order to derive reliable results from DAM, there is need for identifying the number of defects exactly in each phase of software development lifecycle (especially, the number of the generated defects, the number of the detected defects, and the number of the removed defects). However, the E company collects software defect data without classifying the defect origin phase. Thus, if the defect origin phase for software defect data is identified correctly, different results may be derived from DAM. Second, LOC is not well managed in the E company. This causes the problem to calculate the defect measures, such as defect density. If exact LOC is collected, our analysis will produce more reasonable results.

6. Conclusions

Software COQ is one of the most important factors to consider when producing high-quality software consistently in software development organizations. In order to achieve
Identification of Quality Improvement Strategies using COPQ in Software Industry

Effective software COQ management, it is necessary to estimate correct software COPQ and to identify and remove the activities that may cause high software COPQ. Thus, in this paper, we try to examine the possibility of estimating software COPQ in a software development organization that starts establishing software quality improvement strategies, using COQUALMO and DAM that are commonly used as the quality estimation model. In addition, we estimate software COPQ by using the real software defect data. Furthermore, from a long-term point of view, we analyze specifically the variation of software COPQ according to defect removal distribution and software process improvement.

From a practical point of view, in order to estimate software COPQ correctly in a software development organization that starts establishing a software quality strategy, we recommend to establish a defect categorization system first (e.g., Orthogonal Defect Classification (ODC) [10], as a way of categorizing defects found during the software development process and after customer receive). Based on the system, software defect data should be collected in each phase of software development lifecycle, following reliable software measurement processes. Consequently, as appears by Section 4, if these measurement systems are established and conducted in an appropriate way, we can estimate software COPQ more accurately and make use of decision making to improve software quality.

As mentioned in Section 1, the meaning of software COPQ is limited in this paper. However, the E company is going to collect data for external failure cost soon. Thus, after obtaining and analyzing the data, we will estimate original software COPQ (the internal failure cost and the external failure cost) and then will evaluate the result with real software COPQ.

Acknowledgements: This work was supported by the National Research Foundation of Korea Grant funded by the Korean Government (MEST) (NRF-2010-C1AAA001-2010-001014375).

References

Yeong-Seok Seo, Donghwan Shin, Gookhyun Kim, Jongmoon Baik, and Doo-Hwan Bae


Yeong-Seok Seo is a postdoctoral researcher in the Department of Computer Science at Korea Advanced Institute of Science and Technology (KAIST). He received his Ph.D. in Computer Science from KAIST in 2012. His research interests include software cost estimation, software measurement and analysis, mining software repositories, and software process improvement.

Donghwan Shin is a Ph.D. student in the Department of Computer Science at Korea Advanced Institute of Science and Technology (KAIST). He received his Bachelor’s and Master’s degrees in Computer Science from KAIST in 2010 and 2012, respectively. His research interests include software testing coverage criteria, mutation analysis, software COPQ, and safety-critical software.

Gookhyun Kim received his Bachelor’s degree in Computer Science from Soongsil University in 2010 and Master’s degree from KAIST in 2012. He is an investigator of Financial Supervisory Service in South Korea. His research interests include software reliability and optimal resource allocation.

Jongmoon Baik is an associate professor in the Computer Science Department at Korea Advanced Institute of Science and Technology (KAIST). Previously, he was a principal research scientist at Software and Systems Engineering Research Laboratory, Motorola Labs, where he was responsible for leading many software quality improvement initiatives. His research activity and interest are focused on software six-sigma, software reliability and safety, and software process improvement.

Doo-Hwan Bae is a professor in the Department of computer science at Korea Advanced Institute of Science and Technology (KAIST). He received his Ph.D. at the Department of Computer Science in the University of Florida. He currently leads many projects funded by Korean government and industry. His research interests include software safety, software testing, quality driven software development, embedded software design, and mining software repositories.