Scheduling Quality Related Activities in Incremental Software Development Projects Based on Monte Carlo Simulation

Bin Xu, Jie Chen, Yujia Ge, Zhixian Chen and Yun Ling
College of Computer and Information Engineering,
Zhejiang Gongshang University, Hangzhou 310018, China

Abstract: Quality is very important in software development project especially for those related to medical treatment, aerospace, finance and public infrastructure. Quality related activities including feasibility analysis, requirement review, architecture inspection, code review, software testing and auditing run through the entire software development life cycle. Quality related activities are a set of umbrella task corresponding to the defects contained inside the requirement specification, documents and applications. While quality related activity may account for nearly one third or even a half effort and duration in typical software projects, it has been considered as the bottleneck to deliver the software applications in timely model. Based on the effort analysis on the defect finding in incremental software development projects, this paper introduces the effort model of the quality related activities, the calibration based on local historic incremental software projects and detail scheduling method including the efficiency matrix, capability matrix and the definition of the constraints and the goal and suggests scheduling quality related activities while balancing time-to-market and high quality based on Monte Carlo simulation. Pilot in some incremental financial software projects shows that the suggested method will benefit the scheduling of quality related activities in the incremental software development projects.

Key words: Software testing solution, effort optimization, defect removal efficiency, metric calibration

INTRODUCTION

While software applications become large and complex, software quality turns out to be extremely important in most systems especially those related to medical treatment, aerospace, finance and public infrastructure (Xu, 2010, Xu et al., 2004, Vinayagasingkaram and Srivatsa, 2007). Bad quality software applications frequently result in revenue loss, hurts the user confidence, unable to provide competitive advantage to business and may create additional workload (Original Software, 2010). Quality related activity can be regarded as a set of umbrella task corresponding to the defects inside the requirement specification, documents and applications. Quality related activity includes the software quality assurance and software quality control which are defined by CMMI (ISO/IEC, 1998; ISO/IEC 9126-1, 2001; Rose, 2005). It includes feasibility analysis, requirement review, architecture inspection, code review, software testing, auditing and etc., throughout the entire software development life cycle (Shu et al., 2009).

Typically, the scope, deadline, manpower and effort can be negotiated most of the time but software quality is a primary criterion which cannot be jeopardized (Bedi and Gaur, 2007; Bulbul and Sahin, 2007; Ettaiti et al., 2011). However, quality related activity may account for nearly one third to a half effort and duration in typical software projects, it has been considered as the bottleneck to quickly deliver the software applications. A survey conducted by Original Software (2010) revealed that the importance of quality management has risen while most managers are not satisfied with their current quality management solutions.

Our previous research has indicated the most significant defect categories and suggested the priority to improve the quality continuously. When scheduling the quality related activities at the beginning of software projects, we should not only care about the most significant defect categories indicated in the historic projects but also need to deal with several uncertain factors including the staffing, duration and cost. This paper identifies the effort model of the software quality related activities, suggests approach to calibrate the metrics and benchmarks based on local historic software project and balances the time-to-market to quality with Monte Carlo simulation. The efficiency matrix, capability matrix and the definition of the constraints and the goal

Corresponding Author: Bin Xu, College of Computer and Information Engineering, Zhejiang Gongshang University, Hangzhou 310018, China
are defined and several case studies are demonstrated and analyzed. Some increments in a financial software project are chosen as the piloted releases. The Function Point is suggested to size the release on the basis of pilots. The results show that the suggested method will benefit the scheduling of quality related activities in incremental software development projects.

Though there are lots of researches on software quality management, including the total quality management (Deming, 1986; Juran and Gryna, 1993; Crosby, 1979; Chung et al., 2010) unified process (Jacobson et al., 1999; Norberg, 2002; Kroll and Kruchten, 2003; Shuja and Krebs, 2007) six sigma, lean quality management and value-based quality management, few research work is related to scheduling the quality related activities (Xu and Pan, 2006) in incremental software development (Wohlin, 1994).

The authors have done some research on how to continuously improve the quality in the incremental software development projects. In that research, the authors suggested using Pareto analysis to assist the quality improvement process decision by indicating the significant defect categories and root cause (Yu, 1998). When scheduling the quality related activities for a new release, the staff, timeline and budget limitation of quality tasks are non-determined. Such uncertainty results in uncertainty of effort and final quality status. For example, if we arrange too less effort in quality related activities, we could arrange much more effort in development so as to reduce the defect injection but the injected defects may have less chance to be detected. Further, quality related effort distribution in different order will result in different defect removal efficiency. While the project management has budget limitation of the entire project and the request of quality of service, Monte Carlo simulation can be useful for the management to distribute the effort among development and quality tasks and then to distribute both effort at the operational (or detail) level.

Monte Carlo simulations has been widespread used in managing the risks related to different domains, including the construction projects (McCabe, 2003), real-life systems (Kadry, 2007; Abhorr and Olopoe, 2010) investment management (Demir and Bostanci, 2010) wireless network reliability (Peiravi and Hossein, 2008; Rezaii and Tinati, 2009) operational risk access and management (Erugashe, 2009; Xu et al., 2009) Value at Risk computing (Suobokov, 2007).

In software project management domain, Monte Carlo simulation has been used to assess sensitivity in the COCOMO II model for software cost estimation (Musil et al., 2002; Hari and Reddy, 2011) to assess staffing needs in software maintenance projects (Antoniol et al., 2004), to generate the project management solution (Putnam and Myers, 1992).

In this study, the authors will identify the relationship between different activities, calibrate with the local historic quality status and former performance, define the constraints and suggest using Monte Carlo simulation to satisfy the defect removal efficiency request with the constraints of budget limitation based on the activity relationship identification.

**EFFORT MODEL IN SOFTWARE DEVELOPMENT PROJECT**

**Effort dependence between development and quality tasks:**

Though the tasks of a software development project cannot be determined in detail in the early stage, the effort dependence is naturally exists between development tasks and quality tasks. A large scale software project requires more effort for both development and quality tasks. Quality tasks require more ratio of effort in a quality-critical software project while development tasks require more ratio of effort in a technique-critical software project. To be mentioned, insufficient effort in previous stage results in more defects remained and more rework risk in following stages.

**Definitions of development and quality tasks relationship:**

In order to identify the effort model between development and quality tasks, is better to be defined the project, release (or increment), the development tasks, quality tasks and the constraints.

**Definition 1. Project:** A project is defined as PRJ:: = <pid, name>, where pid is the identification number of the project and name refers to the name of the project.

**Definition 2. Release (or Increment):** A release (or increment) is defined as REL:: = <pid, rid, name, size, effort, DRE> where pid is the identification number of the project, rid is the identification number of the release, name, size and effort are the name, functional size and budgeted effort of the release and DRE refers to the defect removal efficiency request.

**Definition 3. Development tasks:** Development tasks refer to all the tasks except the quality related tasks. Here in this research, development task is an abstract concept which refers to all the development other than quality tasks in requirement phase, architecture phase, coding phase and testing phase. A development task can be defined as DT:: = <pid, rid, ph, effort, sdate, edate>, where pid is the identification number of the project, rid is the identification number of the increment, ph is the phase number, effort, sdate and edate are the effort, start date and end date of the development task.
Definition 4. Quality tasks: Quality tasks refer to the quality related tasks. Here in this research, the definition of quality tasks is different than development tasks. A quality task can be defined as $QT := \langle id, pid, rid, ph, tid, effort, sdate, edate \rangle$, where id is the identification number of qt, pid is the identification number of the project, rid is the identification number of the increment, ph is the phase number, tid is identification number of the quality task, effort, sdate and edate are the effort, start date and end date of the quality task. The quality task is categorized as $QTC := \langle tid, name \rangle$, where tid and name are the identification number and the name of the quality task.

Definition 5. Defect list: Defect can be fetched from the defect management system, such as BugZilla, Jira or ClearQuest. A unified definition of defect list can be $DL := \langle id, pid, rid, sdate, rc, pri \rangle$, where id is the identification number of the defect, pid is the identification number of the project, rid is the identification number of the increment, sdate, rc and pri are submit date, root cause and priority of the defect. The root cause is defined as $RC := \langle id, name, labor \rangle$ where id, name and labor refer to the identification number, name and the labor rate of the root cause.

EFORT SCHEDULING USING MONTE CARLO SIMULATION

The entire effort scheduling using Monte Carlo simulation can be divided into calibration, constraints definition, sampling and simulation.

Calibration: Effort is scheduled according to previous project performance and the historic defect status. The effort to detect the defects is an important factor to determine the composition of the quality related tasks and the quality effort distribution between different phases. The capability of the quality related tasks which indicated by the defect removal efficiency is another important factor to forecast the limitation of each quality related tasks composition and the entire defect removal efficiency of the project. The labor cost of the quality related tasks and the cost for the remained defects are the essential impact factors to evaluate the ROI for the effort scheduling.

Most available researches and product (such as COCOMO II and SLIM) towards the software defect estimation suggest that there is linear relationship between defect number and the project size. In this research, the defect number is estimated in linear function on its size. To be more close to the actual value we calculated the optimistic, most likely and pessimistic from the minimal, average and maximal defect density from the historic project. Both Source Lines of Code (SLOC) and Function Point (FP) are used to size the historic projects.

Definition 6. Defect Status Benchmark. Defect status benchmark should be calibrated from found defect list. It can be defined as $DSB := \langle pid, rid, DSL \rangle$, where pid is the identification number of the project, rid is the identification number of the increment and DSL is the defect status list which is queue of $\langle rc, n, wn \rangle$ where $rc$ is the identification number of the root cause, $n$ and $wn$ are the number and weighted number of the defects in the release rid in the category of rc. The defects are weighted by the priority, typically the priority equal to 1 will be weighted as 3, priority equal to 2 or 3 will be weight as 1 and other defects will be weighted as 1/3. $wn$ is calculated as the summary of the defect weight and $n$ is the count of the defects.

Definition 7. Reference project group: Similar projects can be grouped together to serve as the reference in estimating new project. Reference project group is used to distinguish the incremental projects with different profile, performance, or quality status. Reference project group can be defined as $RPG := \langle gid, relSt >$, where gid is the identification number of the reference project group and relSt is the release list which is queue of $\langle pid, rid \rangle$, where pid is the identification number of the project, rid is the identification number of the increment.

Definition 8. Quality task performance: In the execution of each quality related task, defects in different categories may be detected. The performance of quality tasks is defined as $QTP := \langle id, pid, rid, ph, task, TPL \rangle$ where Id is the task identification of qtp, pid is the identification number of the project, rid is the identification number of the increment, ph is the phase id, Task is the task category for quality work and TPL is the queue of $\langle RC, DRE, Effort, Defect Found \rangle$ where RC is the category of root cause, DRE is the defect removal efficiency of Task, Effort is the effort to finish task qtp and Defect found is the number of defects in the RC category found in the execution of qtp.

Algorithm 1: Quality tasks performance calculation

Input: Projects PRJ, releases REL, quality tasks QT, defect list DL, project id pid and release id rid.
Output: quality tasks performance matrix QTP.

1. sort DL in the descending order of pid, rid, sdate
2. FOR (each dl in DL) 
3. IF (dl.pid = pid AND dl.rid = rid) break;
4. submitDate = dl.sdate; defectNumber = 0; defectTotal = 0; effort = 0;
5. FOR (each dl in DL) {
6. IF (dl.pid <> pid OR dl.rid <> rid) break;
7. IF (submitDate - sdate) 
8. defectNumber[rc] ++; defectTotal[rc] ++;
9. ELSE {
10. updateEffort(effort, defectNumber, QT); 
11. generateQTP(pid, rid, effort, defectNumber, QT); 
12. submitDate = submitDate; defectNumber = 0; effort = 0;
13. }
14. } 
15. updateDREinQTP(QTP, pid, rid, defectTotal);
Table 1: Effort distribution for each root cause category

<table>
<thead>
<tr>
<th>Root cause</th>
<th>Defect Number</th>
<th>Effort distribution (man-h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>System design defect</td>
<td>7</td>
<td>$4.2 \times 7 = 29.4$</td>
</tr>
<tr>
<td>Detail design defect</td>
<td>10</td>
<td>$4.2 \times 10 = 42$</td>
</tr>
<tr>
<td>Coding defect</td>
<td>33</td>
<td>$4.2 \times 33 = 138.6$</td>
</tr>
<tr>
<td>Sum</td>
<td>50</td>
<td>210</td>
</tr>
</tbody>
</table>

Fig. 1: Overlapped quality tasks at the timeline

Algorithm 1 walks through defect list of one release, counts the defect for each root cause category and calculates the related effort according to the ratio of defect found, as what is shown in Table 1. The procedure update effort at Line 10 is use to distribute the related effort according to the ratio. When there is overlap of Quality tasks at the timeline, such distribution should consider the ratio of effort of each quality task, as shown in Fig. 1.

The effort is assumed to be level loading, that is, the resource is average staffing for each quality tasks. Therefore, the effort between $t_1$, $t_2$, $t_3$ and $t_4$ will be as formula (1):

$$E_{t_1} = \frac{t_2 - t_1}{t_3 - t_1} \times E_{t_2}$$
$$E_{t_2} = \frac{t_4 - t_1}{t_4 - t_2} \times E_{t_3}$$
$$E_{t_3} = \frac{t_4 - t_2}{t_3 - t_1} \times E_{t_4}$$

After that, the effort can be distributed according to the defect number found as what is demonstrated in Table 1. Since some defects maybe found after the application has been delivered to the clients, such defects will be submitted to the defect management system after the regular quality tasks. Therefore, the final defect removal efficiency could only be calculated after the algorithm 1 walk through all the related defects for the release. The procedure update DRE in QTG at Line 15 in the algorithm 1 is used to update the DRE accordingly:

$$\forall r, DRE_{r} = \frac{DRe_{DRE_{r}}}{DRe_{Defects_{r}}}$$

Most time, the quality tasks are not executed individually. For example, there will be unit testing, code review, functional testing, regression testing and etc. after the code has been developed. It’s hard to evaluate the performance of a single quality task. For example, the unit testing may detect 3 coding defects in 5 h, while at the same time it facilitate the code review and following system testing. Basically, the foregoing tasks brings more benefit than just calculating from the defect found list and the posterior tasks has less benefit than which is calculated Conversely. Before we could find a good model to refine the performance evaluation, quality task group is defined so as to evaluate the quality tasks within a certain phase as a whole.

**Definition 9. Quality task group:** Quality task group is defined as $QTG ::= <tgId, pid, rid, ph, TL, TPL>$ where $tgId$ is the task identification of $qtg$, $pid$ is the identification number of the project, $rid$ is the identification number of the increment, $ph$ is the phase id, $TL$ is the queue of task and TPL is the queue of $<RC, DRE, Effort, DefectFound>$ which is similarly defined as Quality Task but here they are the contributes belong to the group of quality tasks $qtg$. Quality task group performance $QTG$ is calculated on the base of quality tasks performance matrix $QTP$ using Algorithm 2.

Algorithm 2: Quality task group performance Calculation


1. sort $QTP$ in the descending order of $pid$, $rid$, $ph$
2. FOR each $qtp$ in $QTP$
3. IF $(qtp.pid = pid \And qtp.rid = rid)$ break;
4. currentPhase = $qtp.ph$; defectNumber = 0; effort = 0; DRE = 0; TaskList = EmptyList;
5. FOR each $qtp$ in $QTP$ {
6. IF $(qtp.pid <> pid \And qtp.rid <> rid)$ break;
7. IF (currentPhase <> $qtp.ph$) {
8. TaskList.add($qtp.Task$);
9. FOR each $tql$ in $qtp.TPL$ {
10. defectNumber[$tql.rc] ++ = $qtp.DefectFound$;
11. DRE[$tql.rc] += $qtp.DRE$;
13. }
14. ELSE {
15. generateQTP($pid$, $rid$, $ph$, TaskList, DRE, Effort, defectNumber); 16. currentPhase = $qtp.ph$; defectNumber = 0; effort = 0; DRE = 0; TaskList = EmptyList;
17. }
18. }
19.

Quality task group performance matrix $QTG$ can be considered as the summary of the quality task performance matrix $QTP$. Since the defect removal efficiency is an absolute value, here in Algorithm 2, the DRE can be directly summed together to generate the group value:

$$\forall pid, rid, ph, QTP_{pid, rid, ph} = \sum_{r} QTP_{pid, rid, ph, r}$$ (3)

In this research, quality task group performance matrix is actually used in the Monte Carlo simulation but
the quality task performance matrix is still waiting for the future exposure.

**Goal and constraints definitions:** The goal is to achieve the minimal total labor cost for the quality tasks and the cost for remained defects.

\[
\text{Goal:} \min \sum \text{Effort}_{r,ph} * \text{LaborRate} + \sum \text{Defect Remained}_r * \text{penaltyCost}
\]  

\[(4)\]

There are several constraints in the quality effort scheduling, including the budget constraint and capability constraint. Because there is project budget limitation, the quality related activities are limited within the entire budget boundary. The summary of the quality related activities could not beyond a certain part of the budget:

\[
\forall r, \sum \text{Effort}_{r,ph} \leq \text{EffortAvailable}_{ph}
\]  

\[(5)\]

We assume that the tasks couldn't detect more defects than the estimated number:

\[
\forall r, \sum \text{DRE}_{r,ph} \leq \text{100\%}
\]  

\[(6)\]

The capability of the quality related tasks shows that each task has its technique limitation in detecting the defects. In other words, the tasks don't require more effort than its request:

\[
\forall r, \text{ph, Effort}_{r,ph} \leq \text{DRE}_{r,ph} * \text{EstimateDefect}_{r,ph} & \text{Effort}_{r,ph} \geq 0
\]  

\[(7)\]

Besides, we suggest an assumption that the quality related tasks don't have tendentiousness to detect only one kind of root cause. Therefore, kinds of defects will be detected by each quality task as it has done in the previous projects:

\[
\forall r, \text{DRE}_{r,ph} = \text{DRE}_{r,ph} \text{if(DRE}_{r,ph} > 0 & \& \text{DRE}_{r,ph} > 0
\]  

\[(8)\]

**Sampling and simulation:** The effort for each phase and each root cause category is the schedule results and is sampling in this research. Considering that there will be multiple quality task groups can be chosen for different phases, we tried to choose different quality task group for each phase in the simulation.

**Algorithm 3: Simulation**

Input: Reference project group RPG, estimated project size, size, and available effort for each phase AvailableEffort[ph].

Sampling: Effort[ph,rc] and the quality task group for each phase, QTP[ph].

Output: required effort for each phase and each root cause Effort[ph,rc].

1. Estimate the defects for each root cause category as defectEstimated[rc].
2. FOR (each ph in phases) /phases from 1 to 3. Requirement, Architecture and Coding
3. Calculate the capability of the QTP towards different rc from RPG:/DRE[ph,rc]
4. Set the goal in the simulation with totalCost(Effort, DefectRemained) .
5. FOR (each ph) set the budget constraints with budgetConstraints(Effort, AvailableEffort);
6. For (each rc) set the defect number constraints with defectConstraints[DRE];
7. For (each rc) FOR (each ph) set the effort constraints with effortConstraints[Effort,DRE,defectEstimated];
8. For (each ph) set the nature constraints with natureConstraints[DRE];
9. Execute the simulator;

The constraints function total cost at Line 4 is calculated with the formula 4. The function input with the effort matrix and defectRemained matrix and output the total cost to serve as the goal of simulation. The procedure budget constraints at Line 5 defines the constraints of budget according to formula 5 which ensures that the result effort will be limited within the boundary of available effort per phase. The procedure defect constraints at Line 6 defines the constraints according to formula 6 to avoid the unreasonable result which has impossible defect number. The procedure effort constraints at Line 7 defines the constraints according to formula 7 to avoid the impossible effort result. The procedure nature constraints at Line 8 defines the constraints according to formula 8 to avoid the tendentiousness.

**CASE STUDY IN GLOBAL IT CORP**

The suggested approach has been piloted in a financial software company. A finished increment has been chosen from an incremental software development project. The team leader showed the pilot team with the delivered size in SLOC but the quality related effort is kept unknown to the pilot team during the pilot. The metrics and benchmark were calibrated from historic projects at first. Table 2 shows the most available quality related tasks in the project in piloting.

**Calibration according to the historic data:** Table 3 shows the root cause category used in the pilot team. The optimistic, most likely and pessimistic defect density are calibrated with the minimal, average and maximal defect
density of the preceding increments. The unit is in defect per kilo-SLOC or defect per FP.

Table 4 shows the quality tasks that have been performed in the previous increments. Two increments are chosen and they are grouped together into a reference project group to forecast the “new” project (to the pilot team, as they don’t know the actual effort in the pilot). Though the regression testing is determined by the size of application which includes the baseline part and the delivered part, here in this research we don’t distinguish the regression testing from other tests. No special characteristic will be considered for any quality task. Since the business analyst didn’t have their working hours recorded in detail, the business solution document review and architecture review are not mentioned in the first case.

Both releases have two quality tasks, functional testing is to ensure the accurate implementation of the new functionality and regression testing is to ensure there is no defect has been brought for the available functionality.

Table 5 shows the performance matrix for individual quality task. The functional test is more effective and more efficient than the regression test in finding the different root causes. The effectiveness is indicated by the high defect removal efficiency and can be owned to the order of implementation. The efficiency is indicated by the effort cost for each defect which can be owned to the capability and performance of the individual quality related task.

Quality task group performance matrix is the summary of quality task performance matrix, Table 6 summarized the data in Table 5 which will be used as the capability and performance benchmark used for the quality effort estimation.

**Scheduling the quality effort sizing with SLOC and FP:**
The pilot team sized the delivered increment both in SLOC and FP (using IFFUG 4.1), the quality related effort is unknown to the pilot team before the sizing has been finished. Table 7 and 8 show the deviation analysis on effort, DRE and defect number. From the result, we may found that Function Point sizing is good at estimate the effort, SLOC sizing is good at estimate the defect number and both are similar when estimating the defect removal efficiency.
Table 6: Quality task group performance matrix

<table>
<thead>
<tr>
<th>Group ID</th>
<th>Task composition</th>
<th>Root cause</th>
<th>Root cause description</th>
<th>DRE (%)</th>
<th>Effort (L.)</th>
<th>Defect number</th>
<th>Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Build testing</td>
<td>Functional test, Regression test</td>
<td>RQ1</td>
<td>Requirement misunderstanding/unclear</td>
<td>90.69</td>
<td>2114.15</td>
<td>156</td>
<td>4-Build</td>
</tr>
<tr>
<td>Build testing</td>
<td>Functional test, Regression test</td>
<td>RQ2</td>
<td>Requirement change/enhancement</td>
<td>65.91</td>
<td>1698.82</td>
<td>116</td>
<td>4-Build</td>
</tr>
<tr>
<td>Build testing</td>
<td>Functional test, Regression test</td>
<td>SD</td>
<td>System design</td>
<td>100.00</td>
<td>68.49</td>
<td>5</td>
<td>4-Build</td>
</tr>
<tr>
<td>Build testing</td>
<td>Functional test, Regression test</td>
<td>DD</td>
<td>Detail design</td>
<td>89.88</td>
<td>2294.21</td>
<td>222</td>
<td>4-Build</td>
</tr>
<tr>
<td>Build testing</td>
<td>Functional test, Regression test</td>
<td>CD</td>
<td>Coding defect</td>
<td>85.88</td>
<td>1886.58</td>
<td>219</td>
<td>4-Build</td>
</tr>
<tr>
<td>Build testing</td>
<td>Functional test, Regression test</td>
<td>SW</td>
<td>Software System tools</td>
<td>100.00</td>
<td>66.81</td>
<td>6</td>
<td>4-Build</td>
</tr>
<tr>
<td>Build testing</td>
<td>Functional test, Regression test</td>
<td>TC</td>
<td>Test cycle</td>
<td>97.10</td>
<td>543.43</td>
<td>67</td>
<td>4-Build</td>
</tr>
<tr>
<td>Build testing</td>
<td>Functional test, Regression test</td>
<td>CU</td>
<td>Cause unknown</td>
<td>80.00</td>
<td>140.90</td>
<td>8</td>
<td>4-Build</td>
</tr>
</tbody>
</table>

Table 7: Deviation analysis on effort and DRE

<table>
<thead>
<tr>
<th>Pilot</th>
<th>Total effort needed</th>
<th>Total DRE achieved</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Estimated (%)</td>
<td>Actual (%)</td>
</tr>
<tr>
<td>SLOC Pilot</td>
<td>1406 (1094 ~ 1721)</td>
<td>1708</td>
</tr>
<tr>
<td>FP Pilot</td>
<td>1728 (965 ~ 2374)</td>
<td>1708</td>
</tr>
</tbody>
</table>

Table 8: Deviation analysis on defect number

<table>
<thead>
<tr>
<th>Pilot</th>
<th>Total defects</th>
<th>Found defects</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Estimated (%)</td>
<td>Actual (%)</td>
</tr>
<tr>
<td>SLOC Pilot</td>
<td>143 (109 ~ 177)</td>
<td>126</td>
</tr>
<tr>
<td>FP Pilot</td>
<td>178 (112 ~ 242)</td>
<td>126</td>
</tr>
</tbody>
</table>

**DISCUSSION**

While many software metrics have been developed to verify the quality traceability (Singh and Saha, 2010; Xu, 2005), the efficiency of object-oriented technique (Parthasarathy and Anbazhagan, 2006; Boroni and Clausse, 2011; Changchien et al., 2002) and to validate special applications (Pan and Xu, 2010; Vinayagarasundaram and Srivasta, 2007; Bedi and Gaur, 2007) individual metrics makes less sense to determine the status of quality. In our previous research we suggested using Pareto analysis method (Barro, 2009; Zhihuan et al., 2010; Xu et al., 2012) to identify the most significant defect which enables the managers to improve the quality by deal with the most significant root causes. When there is relationship between different defect category, or the related implementation cost to deal with the root cause is not linear with the amount of defect in number, Monte Carlo simulation is better to be adopted so as to generate an nearly-best solution for the quality improvement. It's very important to use such approach in the planning of quality tasks when there are many uncertain factors (such as the resource, cost and staff) with different relationship between them.

In this research, we used Monte Carlo simulation (Mooney, 1997; Elalfy et al., 2004) in the pilot and found that the result is acceptable. Besides, we found that the Function Point (ISO/IEC 20926, 2003) sizing method is better than SLOC sizing method in estimating the quality related effort. This is really good news for the software practice as the final SLOC is rather difficult for us to estimate at the beginning of a software project but we could get the Function Point sizing data if there is sufficient requirement material. The results showed that the gap between estimated effort and actual effort is small but the number of defects in different categories has different gap. The interview with the project team showed that the root cause may be defined in different criteria which results in different ratio in the releases. Such deviation can be reduced when more historic data is calibrated and put into the benchmark.

**CONCLUSION**

Considering the difficulty in making the best decision for quality improvement especially when scheduling the quality effort at the very beginning, here in this paper we suggested using Monte Carlo simulation based approach to identify the uncertainty and relationship, calibrate the historic project data and generate the nearly-best solution for the quality effort scheduling. It has been piloted in an incremental development project. In the pilots, we found that the results may indicate some better alternative effort scheduling solutions than manually scheduling. Though the quality of historic data will result in some deviation in the scheduling, such as the gap for defect estimation in root cause category such deviation will be reduced when there is sufficient historic data to serve as the benchmark.

**ACKNOWLEDGMENT**

This study was financial supported by The Science and Technology Department of Zhejiang Province, China with No. 2008C11009 to Dr. Bin Xu and Education Department of Zhejiang Province with No. yb07035 to Prof. Yun Ling and No. 20061085 to Dr. Bin Xu.
This study was also supported by Zhejiang Gongshang University, China with No. Xgz1102 to Dr. Weigang Chen.

REFERENCES


