Optimizing Test Process Action Plans by Simulated Defect Removal Cost Savings

LJUBOMIR LAZIĆ
Technical Faculty, State University of Novi Pazar
Vuka Karadžića bb, 36 300 Novi Pazar,
SERBIA
llazic@np.ac.yu, http://www.np.ac.yu

NIKOS E MASTORAKIS
Technical University of Sofia,
English Language Faculty of Engineering
Industrial Engineering
Sofia 1000
Sofia
BULGARIA
http://www.wseas.org/mastorakis

Abstract: - To enable software designers to achieve a higher quality for their design, a better insight into quality predictions for their design choices, test plans improvement using Simulated Defect Removal Cost Savings model is offered. In this paper we propose a model which enables to minimize the cost of switching between test plan alternatives, when the current choice cannot fulfill the quality constraints. The defect containment measure is traditionally used to provide insight into project success (or lack thereof) at capturing defects early in the project life cycle, i.e., the time when defect repair costs are at their minimum. We offered a simulation method with which it is possible to assist the test manager in evaluating test plan alternatives and adjusting test process improvement decisions in a systematic manner. Using raw defect containment data and deriving Quantitative Defect Management (QDM) measures early in the development life cycle provides opportunities for a project to identify issues in defect capture before costs spiral out of control and schedule delays ensue. We simulated two scenarios (a standard quality assurance activities plan and comprehensive quality assurance activities plan) and calculated test process improvement potential according to derived QDM measures.

Key-Words: - Testing optimization, Cost of testing Simulation, Test management, Defect Removal Cost

1 Introduction
The importance of software testing has been emphasized more and more, as the quality of software affects its benefit to companies significantly [1-4]. Our research [5,6] concluded that existing approaches for assessing and improving the degree of early and cost-effective software fault detection are not satisfactory since they can cause counter-productive behavior. An approach that more adequately considers the cost-efficiency aspects of software fault detection is required.

Avoidable rework consumes a large part of development projects, i.e. 20-80 percent depending on the maturity of the organization and the complexity of the products [9]. High amounts of avoidable rework commonly occur when having many faults left to correct in late stages of a project. In fact, research studies indicate that the cost of rework could be decreased by up to 50 percent by finding more faults earlier [2, 5, 9]. It might appear easy to reduce the amount of rework just by putting more focus on early verification activities, e.g. reviews and inspections. Therefore, the interest from industry to improve this area is large. To enable software designers to achieve a higher quality for their design, a better insight into quality predictions for their design choices, test plans improvement using Simulated Defect Removal Cost Savings
model is offered in this paper. The model which enables to minimize the cost of switching between test plan alternatives, when the current choice cannot fulfill the quality constraints is proposed. The defect containment measure is traditionally used to provide insight into project success (or lack thereof) at capturing defects early in the project life cycle, i.e., the time when defect repair costs are at their minimum. Much rather we aim to define a simulation method with which it is possible to assist the test manager in evaluating test plan alternatives and adjusting test process improvement decisions in a systematic manner. Using raw defect containment data and deriving Quantitative Defect Management (QDM) measures early in the development life cycle provides opportunities for a project to identify issues in defect capture before costs spiral out of control and schedule delays ensue. In this paper we motivate the need for value-based testing, describe practices supporting the management of value-based testing, outline a framework for value-based test management, and illustrate the framework with examples. We simulated two scenarios (a standard quality assurance activities plan and comprehensive quality assurance activities plan) and calculated test process improvement potential according to derived QDM measures.

2 Quantitative Defect Management

Many companies employ a defect containment strategy in an attempt to reduce software costs and increase software quality. Programs and/or organizations may provide monthly resulting measures from this strategy as part of their team feedback or management reviews. Defect containment divides the engineering development cycle into separate stages and maps the stage in which a defect originated to the stage in which the defect was detected. Defects may originate at any stage of the software development life cycle (although usually the greatest percentage of defects originates in the code and unit test stage). Defects detected in-stage are typically those defects detected during peer reviews or unit tests. Defects detected out-of-stage are those detected after the work product (e.g., design specification, or code) has been delivered to a downstream user (e.g., design released to development team or code released to software integration team). In-stage defects appear along the diagonal cells, as shown in Fig. 4, from hypothetical small project with 100 errors [1], where 13 defects originated and were detected during code and unit test. Out of-stage defects appear in the cells above the diagonal. (In Fig. 4 defects originated in code and unit test, but were not detected until software integration.)

These defect data provide insights to identify which processes cause the most defects and which processes allow defects to escape. Defect containment is usually reported as percentages of defects captured in the stage in which they originated (see Fig. 6) from another large project data history [2]. According to the [2,5,7], a bug that costs 1 cu (cost unit) to fix on the programmer’s desktop costs 100 cu to fix once it is incorporated into a complete program, and many thousands of cu’s if it is identified after the software has been deployed in the field [10], as shown on Fig. 1. Barry Boehm has published several studies [7] over nearly three decades that demonstrate how the cost for removing a software defect grows exponentially for each downstream phase of the development lifecycle in which it remains undiscovered.

![Fig. 1 Engineering Rules for Cost Of Defect Removal][10]

Therefore, there is a large interest in approaches that can reduce the cost of rework, e.g. make sure that more faults are found earlier.

2.1 The defect containment measure

An error in an activity of development phase $P_i$ ($i=1$ to $N$) is made that causes a failure. The failure leads to a reported anomaly. When the reported anomaly is analyzed, the fault(s) causing the failure is found and corrected. Rework is about revising an existing piece of software or related artifact. Therefore, a typical rework activity is to correct reported anomalies. Rework can be divided into two primary types of corrective work [9]:

- **Avoidable rework** is work that would not have been needed if the previous work would have been correct, complete, and consistent. Such rework consists of the effort spent on fixing software
difficulties that could have been discovered earlier or avoided altogether [2,5].

• Unavoidable rework is work that could not have been avoided because the developers were not aware of or could not foresee the change when developing the software, e.g. changed user requirements or environmental constraints [9].

Using raw defect containment data and deriving QDM measures early in the development life cycle provides opportunities for a project to identify issues in defect capture before costs spiral out of control and schedule delays ensue.

2.2 The raw defect containment data
This section is dedicated to a model for assessing a plan for SQA defect-removal effectiveness and cost. The model, a multiple filtering model, is based on data acquired from a survey of defect origins, percentages of defect removal achieved by various quality assurance activities, and the defect-removal costs incurred at the various development phases. The model enables quantitative comparison of quality assurance policies as realized in quality assurance plans. The application of the proposed model is based on three types of data, described under the following headings from [1].

2.2.1 Defect removal effectiveness
It is assumed that any quality assurance activity filters (screens) a certain percentage of existing defects. It should be noted that in most cases, the percentage of removed defects is somewhat lower than the percentage of detected defects as some corrections (about 10% according to [4]) are ineffective or inadequate. The remaining defects, those undetected and uncorrected, are passed to successive development phases. The next quality assurance activity applied confronts a combination of defects: those remaining after previous quality assurance activities together with “new” defects, created in the current development phase. The main objective of the case study presented in this section was to investigate how fault statistics could be used for removing unnecessary rework in the software development process. This was achieved through a measure called Faults-Slip-Through (FST) [5,9], i.e. the measure tells which faults that would have been more cost-effective to find in earlier phases. From the measure, the improvement potential of different parts of the development process is estimated by calculating the cost of the faults that slipped through the phase where they should have been found (see Fig. 7 and 8 below). The usefulness of the method was demonstrated by applying it on two completed development projects [1] and [2]. The results determined that the implementation phase had the largest improvement potential since it caused the largest FST cost to later phases, i.e. from 56 to 87 percent of the total improvement potential in the two studied project scenarios. It is assumed that the filtering effectiveness of accumulated defects of each quality assurance activity is not less than 40% (i.e., an activity removes at least 40% of the incoming defects). Typical average defect filtering effectiveness rates for the various quality assurance activities, by development phase, based on Boehm [7] and Jones [4], are listed in Table 1.

2.2.2 Cost of defect removal
Data collected about development project costs show that the cost of removal of detected defects varies by development phase, while costs rise substantially as the development process proceeds. For example, removal of a design defect detected in the design phase may require an investment of 2.5 working days; removal of the same defect may require 40 working days during the acceptance tests. Several surveys carried out by IBM, TRW, GTE, Boehm and others, summarized by Boehm [7], estimate the relative costs of correcting errors at each development phase. Estimates of effectiveness of software quality assurance tools and relative costs of defect removal are provided by McConnell [10]. Although defect removal data are quite rare, professionals agree that the proportional costs of defect removal have remained constant since the surveys conducted in the 1970s and 1980s. Instead of average per phase defect removal cost we propose average relative defect-removal costs injected in phase $P_i$ ($i=x$ to 7), and detected and removed latter in downstream phases $P_j$, $j>i$, up to the operation phase ($j=7$) as shown in Table 2.

Table 1 Average filtering (defect removal) effectiveness by Standard quality assurance activities plan [1]

<table>
<thead>
<tr>
<th>No.</th>
<th>Quality assurance activity</th>
<th>Defect removal effectiveness</th>
<th>Cost of removing a detected defect (cost units)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Requirement specification review</td>
<td>95%</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Design review</td>
<td>95%</td>
<td>2.5</td>
</tr>
<tr>
<td>3</td>
<td>Unit test - code</td>
<td>95%</td>
<td>6.5</td>
</tr>
<tr>
<td>4</td>
<td>Integration test</td>
<td>95%</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>Documentation review</td>
<td>95%</td>
<td>16</td>
</tr>
<tr>
<td>6</td>
<td>System test</td>
<td>95%</td>
<td>40</td>
</tr>
<tr>
<td>7</td>
<td>Operation phase</td>
<td>100%</td>
<td>110</td>
</tr>
</tbody>
</table>

2.2.2 Quantitative Defect Removal Model
The model is based on the following assumptions:
The development process is linear and sequential, following the waterfall model of CMM Level 5. Software size is approximately 100FP (1 injected defect/FP) i.e. for Java implementation about 50KLOC of source code [4].

A number of “new” defects are introduced in each development phase. For their distributions, see Fig. 3.

Review and test software quality assurance activities serve as filters, removing a percentage of the entering defects and letting the rest pass to the next development phase. For example, if the number of incoming defects is 30, and the filtering efficiency is 60%, then 18 defects will be removed, while 12 defects will remain and pass to be detected by the next quality assurance activity. Typical filtering effectiveness rates for the Standard quality assurance activities are shown in Table 1.

At each phase, the incoming defects are the sum of defects not removed by the former quality assurance activity together with the “new” defects introduced (created) in the current development phase.

The cost of defect removal is calculated for each quality assurance activity by multiplying the number of defects removed by the relative cost of removing a defect (see Table 2, 3rd column).

The remaining defects, unfortunately passed to the customer, will be detected by him or her. In these circumstances, full removal entails the heaviest of defect-removal costs. In this model, each of the quality assurance activities is represented by a filter unit, as shown for Design in Fig. 2. The model presents the following quantities:

- POD = Phase Originated Defects (from Fig. 3)
- PD = Passed Defects (from former phase or former quality assurance activity)
- %FE = % of Filtering Effectiveness (also termed % screening effectiveness) (from Table 1)
- RD = Removed Defects
- CDR = Average Cost of Defect Removal (from Table 1)
- TRC = Total Removal Cost: TRC = RD ×CDR.

The illustration in Fig. 3 of the model applies to a standard quality assurance plan (“standard defects filtering system”) that is composed of six quality assurance activities (six filters), as shown in Table 1.

Table 2 Representative average relative defect-removal costs and fixing multiplier because FST

<table>
<thead>
<tr>
<th>No.</th>
<th>Quality assurance activity</th>
<th>Average Cost of fixing defects</th>
<th>Fixing multiplier (Cost ratio)</th>
<th>Fixing multiplier (Cost ratio)</th>
<th>Fixing multiplier (Cost ratio)</th>
<th>Fixing multiplier (Cost ratio)</th>
<th>Fixing multiplier (Cost ratio)</th>
<th>Fixing multiplier (Cost ratio)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Requirement specification</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Design review</td>
<td>2.5</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Unit test – code</td>
<td>6.5</td>
<td>10</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>Integration test</td>
<td>16</td>
<td>50</td>
<td>10</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>Documentation review</td>
<td>16</td>
<td>110</td>
<td>26</td>
<td>13</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>System test</td>
<td>40</td>
<td>368</td>
<td>64</td>
<td>37</td>
<td>7</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>Operation phase</td>
<td>110</td>
<td>400</td>
<td>75</td>
<td>40</td>
<td>20</td>
<td>15</td>
<td>10</td>
</tr>
</tbody>
</table>

A comprehensive quality assurance plan (“comprehensive defects filtering system”) achieves the following: (1) Adds two quality assurance activities, so that the two are performed in the design phase as well as in the coding phase.

(2) Improves the “filtering” effectiveness of other quality assurance activities.

![Fig. 2 A filter unit for defect-removal effectiveness: example (100 defects) from [1]](image)

The comprehensive quality assurance plan can be characterized as shown in Table 3.

The main conclusions drawn from the comparison are:

(1) The standard plan successfully removes only 57.6% (28.8 defects out of 50) of the defects originated in the requirements and design phase, compared to 92.0% (46 defects out of 50) for the comprehensive plan, before coding begins.

(2) The comprehensive plan, as a whole, is much more economical than the standard plan as it saves 41% of total resources invested in defect removal, compared to the standard plan.
Compared to the standard plan, the comprehensive plan makes a greater contribution to customer satisfaction by drastically reducing the rate of defects detected during regular operations (from 6.9% to 3%).

The comparison also supports the belief that additional investments in quality assurance activities yield substantial savings in defect removal costs. Alternative models dealing with the cumulative effects of several quality assurance activities are discussed by [2, 5, 9] as described below. A process-oriented illustration of the comprehensive quality assurance plan and model of the process of removing 100 defects is provided in Fig. 4. A comparison of the outcomes of the standard software quality plan versus the comprehensive plan is revealing as shown in Table 3.

### Table 3 Comprehensive quality assurance plan [1]

<table>
<thead>
<tr>
<th>No.</th>
<th>Quality assurance activity</th>
<th>Defect-removal effectiveness</th>
<th>Cost of removing a detected defect (cost units)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Requirement specification review</td>
<td>60%</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Design inspection</td>
<td>2%</td>
<td>2.5</td>
</tr>
<tr>
<td>3</td>
<td>Design review</td>
<td>60%</td>
<td>2.5</td>
</tr>
<tr>
<td>4</td>
<td>Code inspection</td>
<td>7%</td>
<td>6.5</td>
</tr>
<tr>
<td>5</td>
<td>Unit test - code</td>
<td>60%</td>
<td>6.5</td>
</tr>
<tr>
<td>6</td>
<td>Integration test</td>
<td>60%</td>
<td>16</td>
</tr>
<tr>
<td>7</td>
<td>Documentation review</td>
<td>60%</td>
<td>36</td>
</tr>
<tr>
<td>8</td>
<td>System test</td>
<td>60%</td>
<td>40</td>
</tr>
<tr>
<td>9</td>
<td>Operation phase</td>
<td>100%</td>
<td>110</td>
</tr>
</tbody>
</table>

### 3 Simulation results of QDM improvement

Unlike conventional approaches to software testing which are applied to the software under test without an explicit optimization goal, as described above, the **IOSTP with embedded Risk Based Optimized STP (RBOSTP)** approach designs an optimal testing strategy to achieve an explicit optimization goal, given a priori [5, 6].

Fig. 4 Defect removal effectiveness of Comprehensive quality assurance plan and model of the process of removing 100 defects [1]

Improvement of original project data from [2] given in Fig. 5 (Note: original Defect Removal Efficiency [%], shown on Fig. 5 is less than Standard quality assurance activities plan (Scenario 1) and comprehensive quality assurance plan (Scenario 2) is realised through feasible series of experiments: software test method, field test, through simulation, or through a combination, which represent test scenario (in our case Scenario 1 with and 2) i.e. sequence of test events.

Fig. 3 Defect removal effectiveness and costs of standard quality assurance plan and model of the process of removing 100 defects [1]
Table 4 Comparison of the standard and comprehensive quality assurance plans

Net savings are calculated using this formula:

\[ NS = FST_{r \rightarrow p+1} \times (CM_{r \rightarrow p+1} - CM_{r \rightarrow p}), \ r = 1..6 \]

for the given large (~11300 FP, Java implementation about 600KLOC of source code) project example from [2], for original data of process defect removal effectiveness is given in Fig. 5, and two Scenarios 1 and 2 are shown in Fig. 7 and Fig. 8.

Fig. 5 Original Software Process Defect Containment Matrix [2]

Fig. 6 Software Defect Containment Percentage Matrix – PCE
4 Conclusion
Testing is one of the most resource-intensive activities in software development and consumes between 30 and 50% of total development costs according to many studies. We described Software Quality Optimization (SQO) strategy which is a continuous, iterative process throughout the application lifecycle resulting in zero-defect software that delivers value from the moment it goes live, with Simulated Defect Removal Cost Savings model. The results determined that the implementation phase had the largest improvement potential since it caused the largest FST cost to later phases, i.e. from 56 to 87 percent of the total
improvement potential in the two studied project scenarios.

References: