28TH ANNUAL
PACIFIC NW
SOFTWARE
QUALITY
CONFERENCE

OCTOBER 18TH – 19TH, 2010

ACHIEVING QUALITY IN A COMPLEX ENVIRONMENT

Conference Paper Excerpt from the

CONFERENCE PROCEEDINGS

Software Quality Management System

Omar Alshathry
Software Technology Research Lab
De Montfort University
The Gateway, Leicester LE1 9BH, UK
shathry@dmu.ac.uk

Abstract

In software development projects, the investment of quality improvements needs to be optimized in a way that does not affect the cost and schedule aspects. However, as is currently practiced in the industry, software's artifacts are considered equal in their significance and risk to the software life cycle with respect to quality improvement activities. The investment in activities concerning the detection and removal of defects is distributed evenly on the software's artifacts without taking into consideration the risk and significance factors of such artifacts. Some software's modules hold risky and significant architectural components that need to be of a high quality and a low defect density. On the other hand, other modules do not require a similar level of quality. Defects originating from modules of high criticality may contribute to a project failure more so than less significant modules. In this paper, we propose a model that helps the project managers to optimize the investment given to the QA activities of their software on the basis of the risk associated with the development process.

Keywords: CosQ, Software Quality, Cost of Quality, Return on Investment, Quality Management

1: Introduction

A well known phrase among software quality practitioners is, "too little is a crime, but too much testing is a sin". This statement reflects the ongoing research problem in the Quality Assurance (QA) area when project managers, at the start of or before the testing phase, should make informed decisions to tune and control the triple constraints in a way to assure the success of their software projects **(Figure1**).

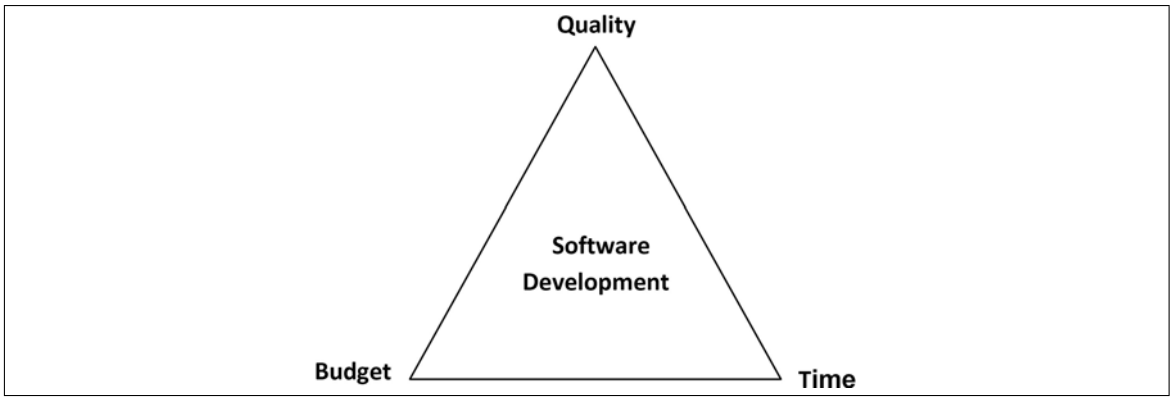

Figure 1: Software Triple Constraints

In some cases, schedule becomes the most important aspect when cost of delay outweighs the benefit of producing software of low defect density. In other cases, a limited budget assigned to the quality plan hampers the quality assurance team from covering software work products sufficiently to detect and remove defects, even those that are deemed to be of a high or moderate risk.

To tackle this issue, we propose a software quality management system that helps the project manager and QA practitioners to make informative decisions on their QA activities in terms of their cost effectiveness and to trade off alternatives of QA plans based on optimal solutions generated. We propose a regression-based generic model that categorizes each each phase's artifacts to different work products according to pre-defined risk levels so as to prioritize investment given to the quality assurance process.

The outline of our paper is as follows. In **Section 2**, we outline the cost of software quality (CoSQ) principal and the cost percentage that cost of software quality consumes out of the total software development budget. In **Section 3**, we outline our quality model informally. The formal descriptions of the model then follows in **Sections 4**. A literature review of similar models is discussed in **Section 5**. We conclude our paper with a critical review of our model in **Section 6**.

This paper is built upon our previous research on risk-based QA activities management [24][27].

2 Software Quality

2.1 Software Quality Perception

In our research, quality of software is intrinsically linked to the notion of software defects and defect density. Software defects can be defined based on different views and perspectives [15][2]. However, it is commonly agreed that a software defect is any flaw present in the software that prevents the normal software execution, causing software failure or nonconformance to software specifications [18]. We chose the defect density metric as our perception of quality and it is calculated as follows :

$$\text{Defect Density} = \frac{No.\ of\ defects\ in\ a\ code}{Size\ of\ the\ code}$$

2.2 Cost of Software Quality

The term cost of software quality *(CoSQ)* describes the trade-off between delivering high quality software and the cost associated with it [1]. It is a metric that helps software project managers to determine the accepted and affordable level of quality for their product.

It is generally accepted that during the software development process, the earlier a software defect is fixed and removed, the more effort and time is saved for the whole project [37] [32]. Software quality researchers estimated that software defect that cost less than $1 to be fixed during software coding phase may cost $100 if it propagated to the whole system and thousands of dollars if it passed to the operational field [38]. This is also supported by Humphrey[23] who showed that the rework cost of a defect found in the field is ten times greater than the cost to remove it during the system testing phase. The rationale behind this escalation in the cost of defects is that to fix a defect which was discovered relatively far from its origin entails not only the cost of fixing it but also the needed re-working[8][10] of other modules impacted by this defect in previous stages **(Figure2**).

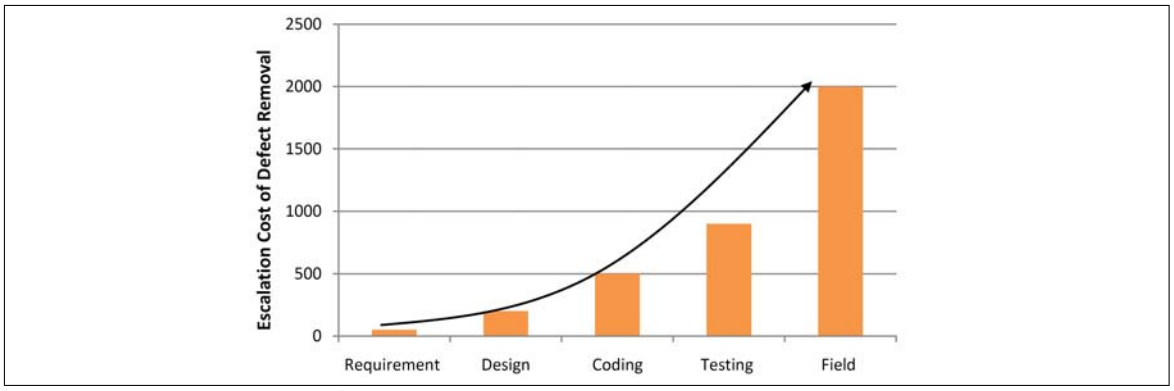

Figure 2: Escalation Cost in Software Defect

2.3 Software Quality and Profit

As shown in **Figure 2.3**, the relationship between software quality and profit is linear at the beginning, as implementing practices and techniques removes defects and improves software quality. However, this linear relationship reverses at one point during the software development process with a decrease in expected profit and a continuous increase in quality. On the other hand, some researches found that Pareto Law, known as the 80:20 rule, is applicable to software quality activities. It was found that 80% of software defects discovered in the system testing phase are related to 20% of the software modules[7][12]. Those two concepts of profit of software quality and the

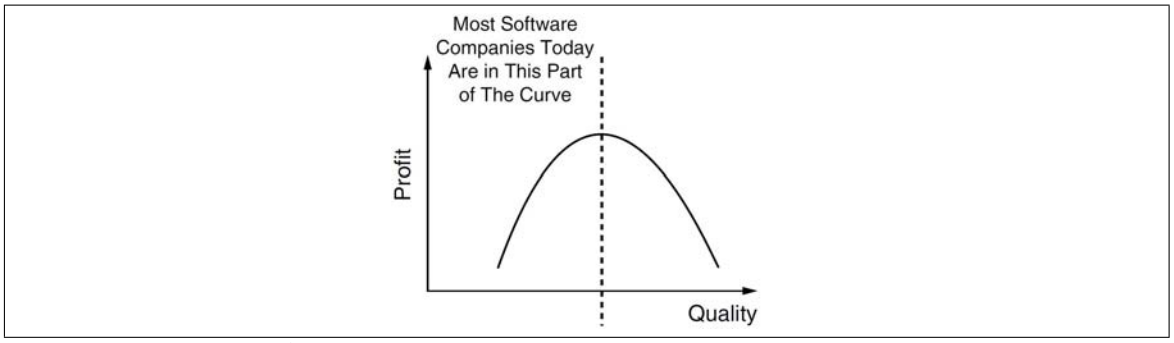

Figure 3: Relationship Between Quality and Profit [5]

Pareto Law are important in any software development project especially when it experiences a shortage in budget assigned to quality or time to release constraints. However, what are the criteria behind the decision of leaving work products uninspected to meet release time targets or re-using a module to save testing cost; how the project manager can handle such issues with minimal risk. In that context, there must be a mechanism that gives the project manager the ability to differentiate between high-risk modules that require more investment in QA activities and low-risk modules.

3: Quality System Model

Our approach to the optimal use of quality assurance practices and performing the required trade-off process between the software triple constraints is regression-based. In other words, we perform a regression analysis on a pool of QA data which was grouped and channelled according to our model specifications. The output of this analysis process will help determine the accurate effectiveness of

QA practices, their cost and time with respect to categorized software work products on the basis of the risk level associated with them. The application of our model conforms to any software development life cycle as long as it consists of phases. As shown in **Figure 4** which describes the work-flow steps of our model, each phase's deliverable will go through a categorization process to categorize each artifact to different work products.

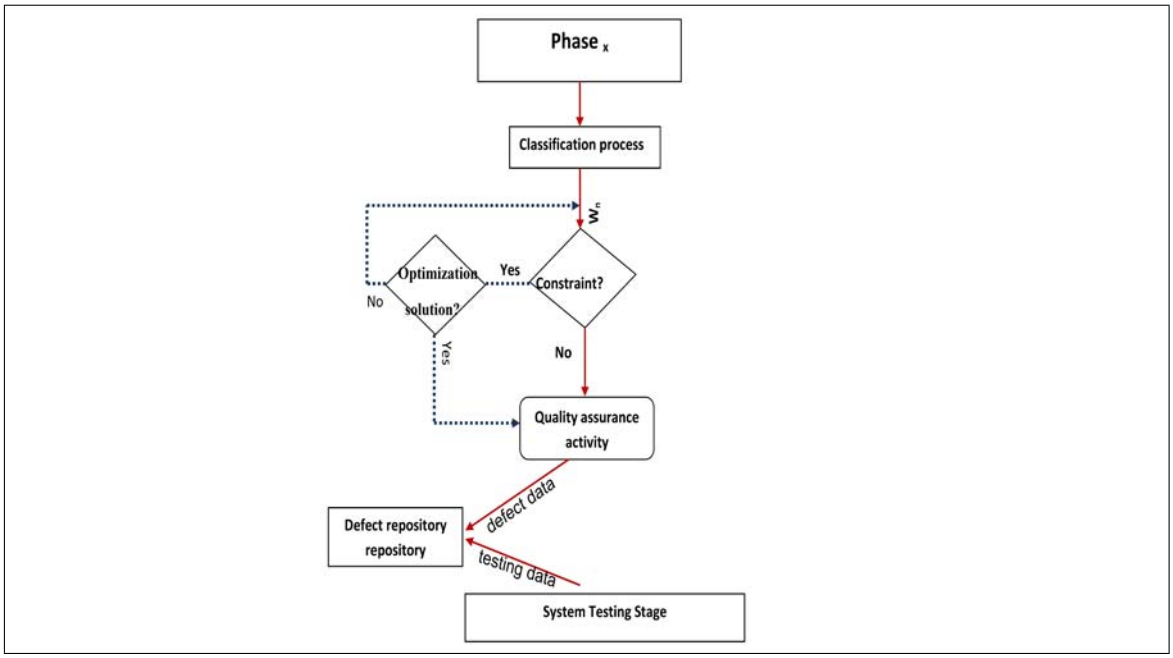

Figure 4: Work-flow of our Quality Model

The next step after the categorization process is the constraint check, which is a decision-based process conducted by either the system's project manager or the system's stakeholders who may put some items and conditions to be considered in the system to be built. Once constraints have been determined they go through the optimized solution determination process. In this process, constraints are grouped together and processed using a suitable optimization technique which will propose an optimal solution for QA plans. The optimal solutions found will be based on two or more of the software triple constraints: cost, quality and schedule. In case there is no optimal solution found, the constraints should be rediscussed between the software stakeholders and the project manager and redesigned for better manipulation of the optimization process. Once optimal solutions are found or in case there were no constraints determined, the QA team should start the QA process using suitable practices. Data of any QA activity in terms of the number of defects found, the duration of executing the QA activity, the average defect removal cost by the QA practice, etc., will be passed to the defect repository process where it is channeled and stored according to the categorized work products.

3.1: Software Models Categorization

The project deliverable of each phase in the SDLC can be broken down into work products. We assume that these work products can be assigned to a domain of specific type and risk categories. For a limited QA planning budget, the highest share of the budget should go to the most critical work products rather than being distributed evenly. For example, in a software requirement specification document requirements are typically ranked based on importance to the client. Here

more effort should be given to the verification of requirements that are important to the user than to those that are in the "waiting room"[11]. Having the risk levels determined, each work product of software development phases is categorized based on a defined set of risk levels derived from the consideration of the risk associated with the software development process. The categorization process is as follows:

Let C be a set of all categories that are used in the domain for which the software is being developed. Every work product can be placed in one category. Let W_x be the set of work products that are generated in phase x. The function $type : W_x \mapsto C$ then assigns for every work product a type category. Multiple categorization schemes can be modelled along these lines. Similarly let $esize : W_x \mapsto N$ be a function mapping from a work product to an estimated size in kLoC. Although for early work products other size measures, such as object or function points, are more suitable, it is well established to translate these into kLoC for the target language of the project [20]. The size $size_x$ of the output of phase$_x$ is then

$$size_x = \sum_{w \in W_x} esize(w)$$

The size of work-products of a specific type $c \in C$ is then:

$$size_{x,c} = \sum_{w \in W_{x,c}} esize(w)$$

where $W_{x,c} = \{w \in W_x | type(w) = c\}$.

The proportion of work products with that are in category c is then:

$$\alpha_{x,c} = \frac{size_{x,c}}{size_x}$$

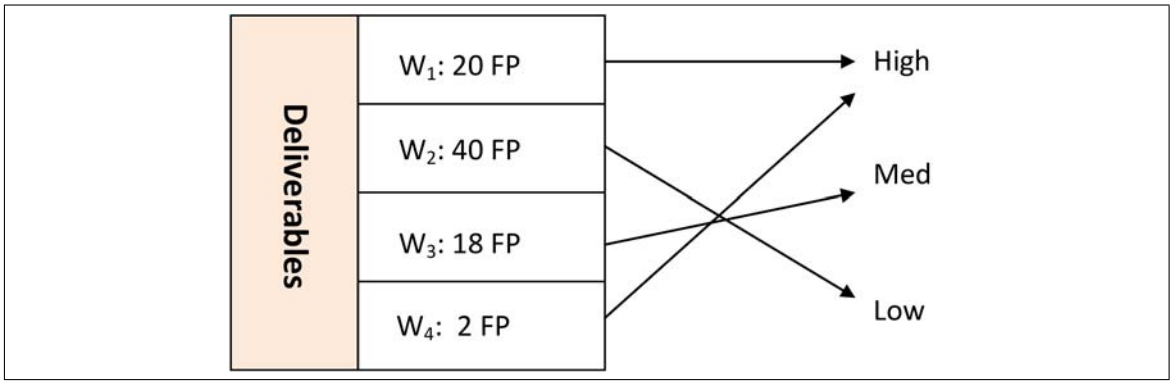

Figure 5: Phase Categorization Process

The categorization of a phase's deliverable work products depends on the project manager insight or the QA team consideration for the deliverable itself and for the type and prospective domain of use for the software. Some deliverable would be given a full weight value $\alpha_{x,high}$ of a high risk rating out of the total deliverable size $size_x$. Conversely, others will be given full $\alpha_{x,low}$ weight of low-risk rating. An example of a mapping from work products to risk categories is shown in **Figure 5**.

3.2: Categorization Scenarios

The process of categorizing software phases into work products is independent of the software size and the risk levels defined. That is, having a set of risk levels does mean that each phase deliverable need to be categorized according to these levels at once. The constraints experienced during software development process and the project manger insight are the main determinant factors for the way this categorization process is implemented. In the following subsections, we will show how this methodology is applied to the software requirement specifications by giving some scenarios of work products categorization.

3.3: 1st Scenario

As an example and as illustrated in **Figure 6**, it shows that a Software Requirements Specification (SRS) document is divided according to our risk-based levels (*high, medium, low*) into three work products of sizes 30%, 50%, 20%, which are inspected by QA practices P_1, P_2 and P_3 respectively. The result of this inspection process is the Defect Removal Efficiency (DRE) value for each QA practice with respect to the type of the work product it is applie to. The DRE value will be known to the QA team at the system testing phase when total defects originated from the work products uncovered.

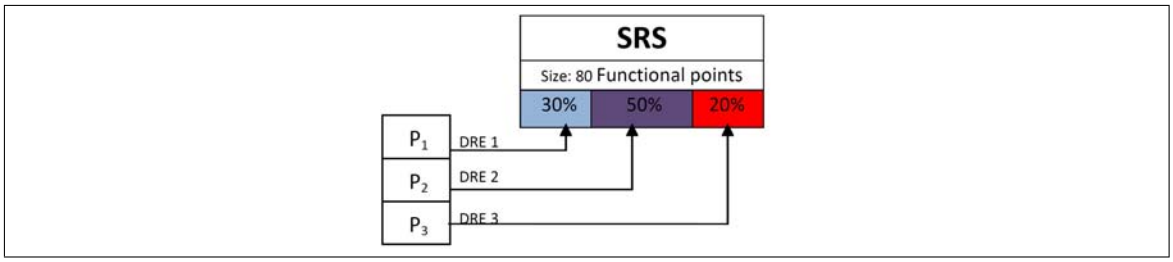

Figure 6: Work Product Categorization (a)

3.4 2nd Scenario

In another scenario, **(Figure 7**), the project manager may have no available budget to inspect all the artifacts or some of the QA practices may only be available for a limited period of time. Accordingly, the QA team manager chose to inspect the high and medium work products only, which is weighted at 50% of the total artifact with the coverage values of 30% and 20% for QA practices P_1 and P_2 respectively, and leave the left 50% of low-risk work product uninspected.

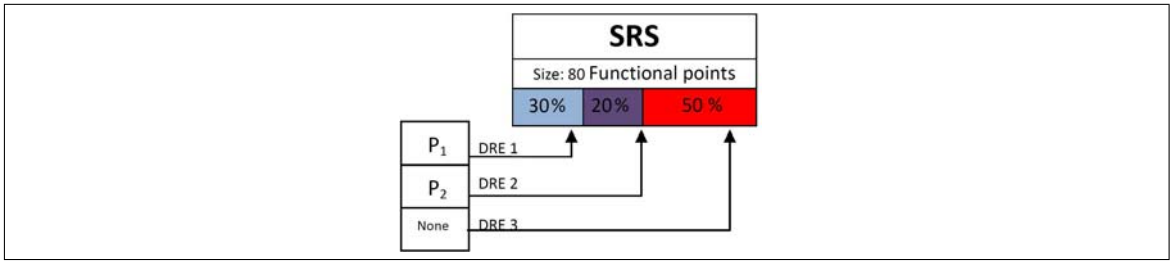

Figure 7: Work Product Categorization (b)

3.5 3rd Scenario

The Categorization process may continue to include the work product itself with respect to the coverage weight given to the QA practices chosen. For example, in **Figure 8** , the project manager may not give practice P_1 a full coverage on the work product of type high because P_1's availability stopped all of a sudden in the middle of the testing or due to other constraint. Accordingly, the project manager used practice P_3 to inspect the remaining part of the work product.

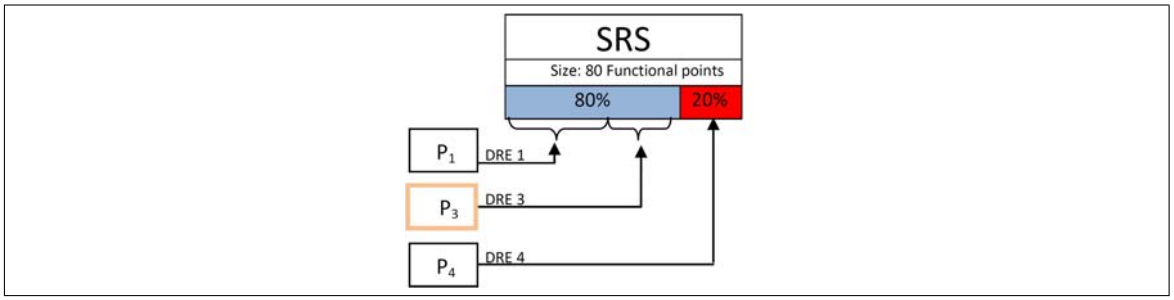

Figure 8: Work Product Categorization (d)

Those three different scenarios mentioned above occur repetitively in software development environment without exploiting them for future use. Software organizations should have a comprehensive and solid background on their QA practices efficiency for each phase and should also have a QA repository of previous projects.

3.6: Data Repository of QA activities

The application of this approach for many projects following the same software development process enables the QA team to build up a data repository of defects reports. In this section, we will show how data resulting from our model can be sorted and processed in a database so that it can be utilized for making decisions for future usage.

After the work products categorisation process, the QA team stores in the data repository all relative data related to each work product like its size out of the total phase size and its risk level **(Table 1)**.

Software Requirement Specification			
Project$_{id}$	Document size (FP)	Type of work product	Work product weight
1	Size$_1$	Type$_1$	$\alpha_1\%$
1	Size$_1$	Type$_2$	$\alpha_2\%$
1	Size$_1$	Type$_3$	$\alpha_3\%$
...	.....	.....	{100%}
2	Size$_2$	Type$_2$	$\alpha_2\%$
2	Size$_2$	Type$_3$	$\alpha_3\%$
3	Size$_3$	Type$_3$	$\alpha_3\%$
4	Size$_4$	Type$_4$	$\alpha_4\%$
5	Size$_5$	Type$_5$	$\alpha_5\%$
.	.	.	.
.	.	.	.
.	.	.	.
n	Size$_n$	Type$_n$	$\alpha_n\%$

Table 1. Work Products Input Table

Having completed the data input process, the data analysis process starts by grouping the dependent variables together for analyzing and determining the relationships between them. The mechanism on which our analysis process is based is the average values of variables and the regression analysis. Our average values determination process will analyze all QA data of past projects that are stored according to our model structure. This process will be applied on the following variables and database tables:

- **Each QA practice with respect to the phase and work product type.**
  The result of this table will determine the average DRE value for a specific QA practice with respect to the work product and to the phase it is applied to.

Phase	Work product	QA practice	$\bar{DRE}$

- **The execution time for each QA practice with respect to the phase and work product type.**
  The result of this analysis process is to determine the average time $\bar{time}$ value needed by QA practice p_1 to run on a work product w_x

Phase	Work product	QA practice	$\bar{time}$

- **Each QA defect removal cost with respect to the phase and work product type.**
  The result of this association will be:

Phase	Work product	QA practice	$\bar{cost}$

To calibrate our model we require a defect injection rate (I) that links the number of defects injected in every work product with its size. Given a positive correlation between the size of work products of a specific type and the number of defects originating from that work product, we can use linear regression to determine the injection rate Ic as the slope of number of defects originating from work products of that type against the size of the work products **(Figure 9)**.

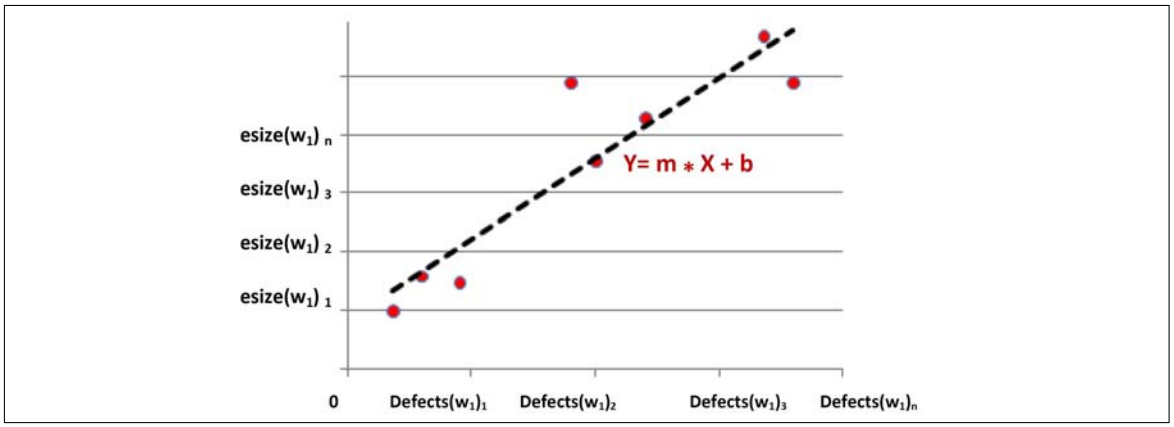

Figure 9: Proposed Regression Analysis

4 Formal Model

Following the SDLC approach being used by the software development organization, our software quality model consists of a sequence of phases: requirement, design and code which are referred to as (r, d, c) respectively. Moreover, each phase consists of a number of work products $w_1 \rightarrow w_n$. On the other hand, there are a number of QA practices which are specifically used and responsible for the defect detection and verification activities within each phase **(Figure10)**.

4.1 Model cost aspects

Generally, the formal model of our system consists of three main components as follows:

1. **Number of Defects (I)**
   This component refers to the estimated number of defects found by applying a specific QA practice to a single work product (w) or to a whole phase in general. The number of defects component of our model will be responsible for determining the DRE value with respect to the QA practice used.
2. **Execution Cost and Effort**
   The execution cost is the cost of performing the QA process using a specific QA practice. As we aim in this research to optimize the cost of QA investment, we need to accurately quantify this value to reduce the waste in effort introduced by any QA activity. In our model, the cost of execution is divided into two blocks:
   - Cost to run the QA practice.
   - Cost to remove defects discovered.

   The reason for this division is that during a QA activity, which comprises defect detection and removal, the defects found or some of them may not get removed during the QA activity even though they were discovered. the QA team may prefer to remove only part of the found defects due to unexpected constraints or because of the fact that the defects found are not necessary to be fixed.
3. **Cost of Escaped Defects**

   In this component, we quantify the impact of defects escaped from the main development phases of the SDLC: Requirement, Design, Coding, etc., with respect to the system testing

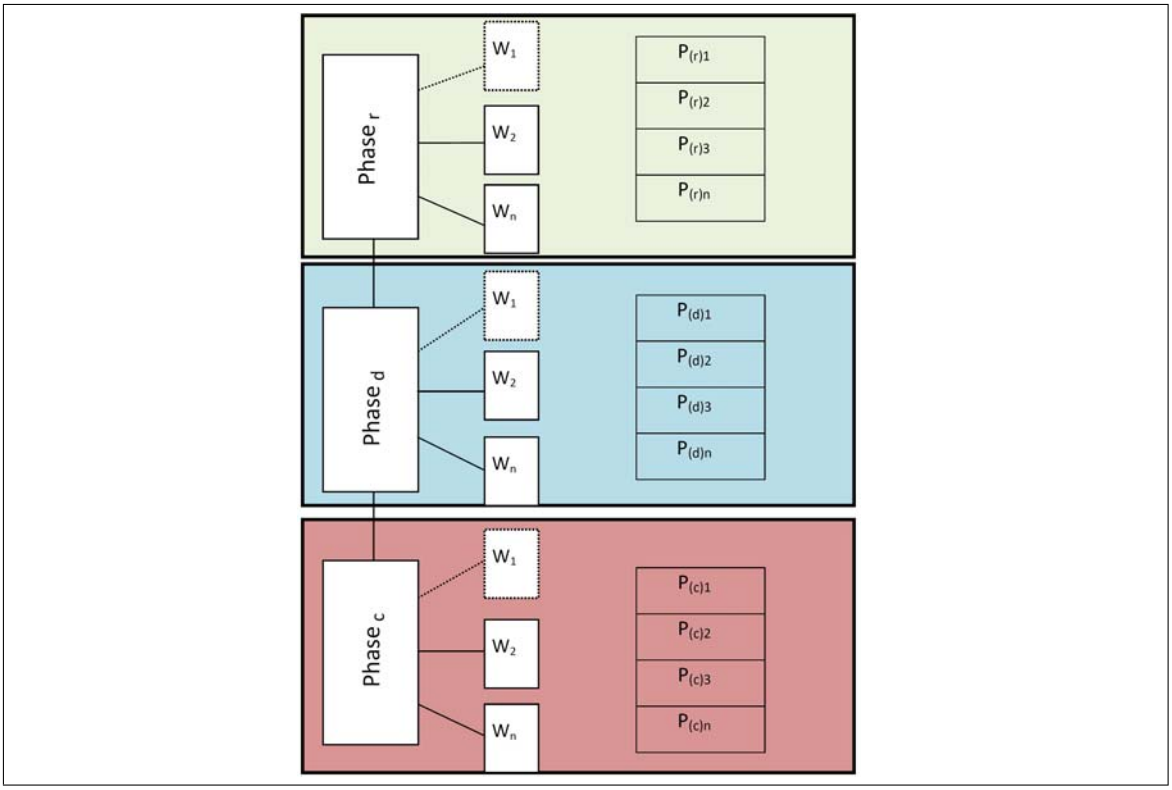

Figure 10: QA Practices and Phases Association

stage. An overview of this association is depicted in **Figure 11**.
The mechanism of our model works by associating QA activities carried out during the development activities of the SDLC's phases with the system testing phase, where the main testing activities begin. This association mechanism should quantify the impact of all escaped defects in terms of the estimated cost associated with their defect removal activities at the system testing phase.
As a result of this process, we will be able to measure the estimated cost introduced by a defect escaped from a work product of a specific risk level and discovered in the system testing phase. We refer to this value in our model as the escalation factor of a defect ($C^{escaped}$).

- P_x denotes the set of all QA practices that can be applied in $Phase_X$.
  Let $p \in P_x$
  Let $w \in W_x$
- Let I_w be the estimated injection rate of defects per kLOC in w of $phase_X$.
- Let β_p refers to the coverage weight of a practice p during a QA activity.
- Let $C^{escaped}$ refers to the escalation factor of an escaped defect with respect to the system testing phase.

1. Estimated Number of Defects Injected

$$eD_w = I_w * esize(w) \tag{1}$$

2. Estimated Number of Found Defects

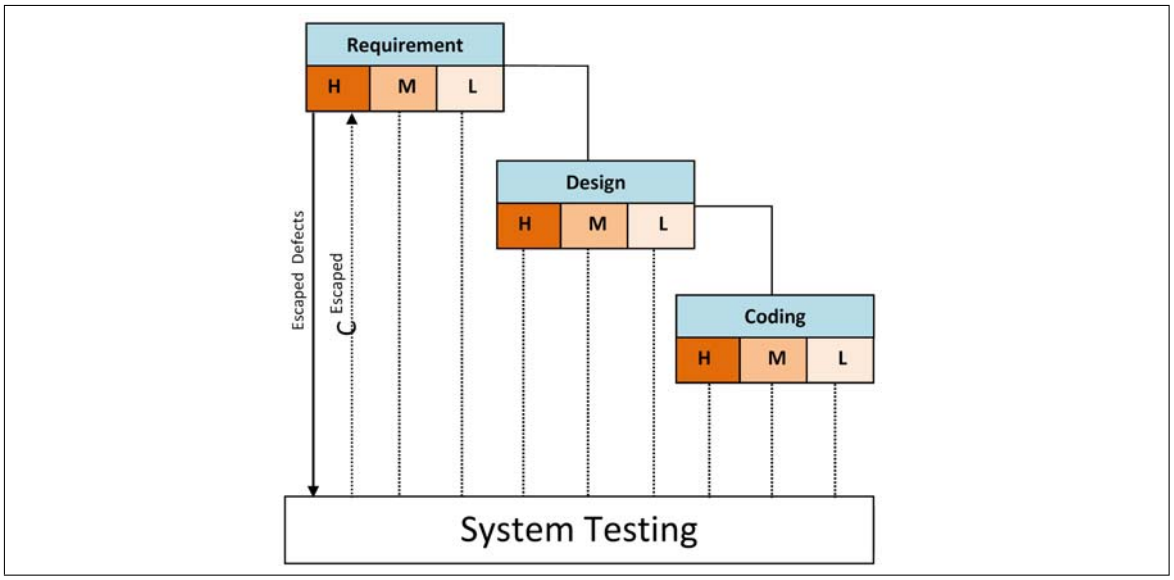

Figure 11: Cost of Escaped Defect

$$N_p^{Found} = \beta_p * eD_w * DRE_p \tag{2}$$

3. Estimated Number of Defects Escaped

$$N_p^{Escaped} = \beta_p * eD_w * (1 - DRE_p) \tag{3}$$

4.2 Execution Time and Effort

4.2.1 Execution Time

The execution time is defined in our model as th time required to run a QA practice on a software's artifact in order to detect and remove its defects. As we mentioned earlier, we need to quantify the cost of running the QA practice as a separate cost from the overall defects removal cost as in many cases software quality practitioners may find defects but not remove them.

- Let t_p be the average execution time of applying a QA practice $p \in$ P on a specific work product $w \in$ W.
- Let size$_w$ be the size of a work product measured based on the artifact's unit of measurement.

Execution time for a single QA practice p

$$Ext_p = \beta_p * size_w * t_p \tag{4}$$

4.2.2 Execution Effort

- Let C^{escaped} be the escalation factor of a defect removed at the system testing phase.
- Let Lr be the labour rate measured in any unit of money.

1. Execution cost for a single QA practice p

$$Exc = Lr * Ext_p \tag{5}$$

2. Cost of Defect Removal Rc

$$Rc_p = \beta_p * eD_w * DRE_p * C_p^{removal} * Lr \tag{6}$$

3. Cost of Escaped Defect Esc

$$Es_p = \beta_p * eD_w * (1 - DRE_p) * C_p^{escaped} * Lr \tag{7}$$

4.3 Saved Cost of a QA Activity

During the process of applying a QA practice, QA practitioners usually do not quantify a major important value that has a great influence in evaluating the current QA plan. This value is defined in our model as the saved cost which refers to the saving in cost of applying a QA practice which is expected to pay off later in the system testing phase.

$$Sc_p = \beta_p * eD_w * DRE_p * C_p^{escaped} \tag{8}$$

4.4 Combination of QA Practices

In some cases and for software artifacts of a high significance, the project manager may like to apply more than one QA practice at once in order to reduce the defects injection rate in later phases.

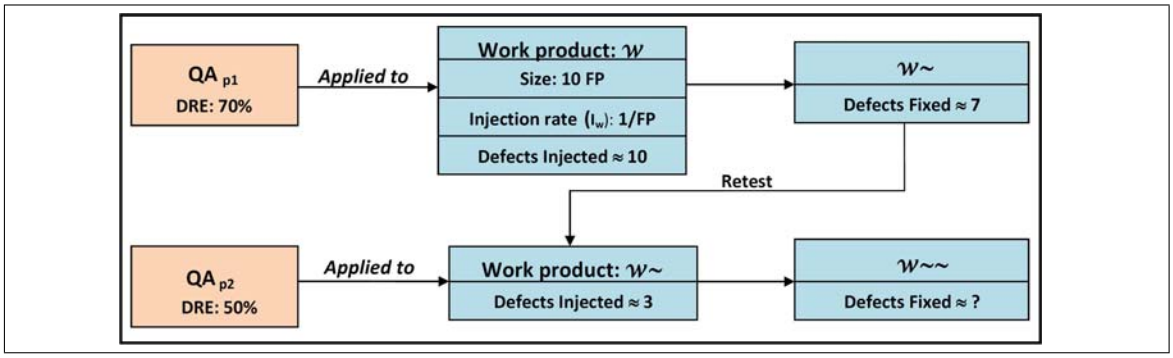

Figure 12: Combining QA practices

In order to clarify the notion of applying more than one QA practice, we give an example as shown in **Figure 12** that depicts a possible defect detection and removal scenario. As an example, the testing team chose to apply QA practice p_1 to a specific work product w that has an estimated injection rate (I_w) of 1/FP, that is, 1 defect is injected into each functional point of the work product. Assuming that the size of the work product is 10 FP, that means the work product is estimated to be injected with 10 software defects. The QA practice p_1 is known to have a defect removal efficiency of DRE = 70% on the work product w relying on data of past projects retreived from the model's repository. The estimated result of this activity was a new tested work product $\bar{w}$ which had approximately 7 defects that were fixed out of the original injected 10 defects.

70% * 10 = 7 defects found.

On the other hand, 3 defects are still injected in the work product $\bar{w}$ which will propagate to the later phases.

(1-70%) * 10 = 3 defects escaped.

In order to reduce the impact of these 3 escaped defects, the testing team decided to retest the same work product $\bar{w}$ using another QA practice p_2 which has a defect removal efficiency value DRE = 50% with respect to such work product type. However, The defect injection rate of work product $\bar{w}$ needs to be redefined according to the result of the first QA activity. We assume that the new injection rate of work product $\bar{w}$ is 0.3 FP as follows:

$$I_{\bar{w}} = \frac{Defect\ injected \approx 3}{Size : 10\ FP} = 0.3$$

Based on the values of $I_{\bar{w}}$ and the DRE of the QA practice $p_2 = 50\%$, the testing team would expect that the verification process would be estimated to reveal half of the injected defects. However, this optimisitic view of the second re-testing process may not always be correct; re-testing the work product with the QA practice p_2 may find no defects despite its defect removal efficiency value of DRE = 50%. The reason behind that is that in some cases the type of defects found by practice p_1 are the same defects found by practice p_2; therfore, applying practice p_2 on a work product which was already tested by practice p_1 may consume effort and time without tangible benefits.

This potential scenario is well-thought about in our research and we tried to tackle it by devising a new variable to calculate the probability that a QA practice p_2 will find defects different to those defects found by the preceding QA practice p_1. We call this variable λ. An overview of this variable is depicted in **Figure 13**.

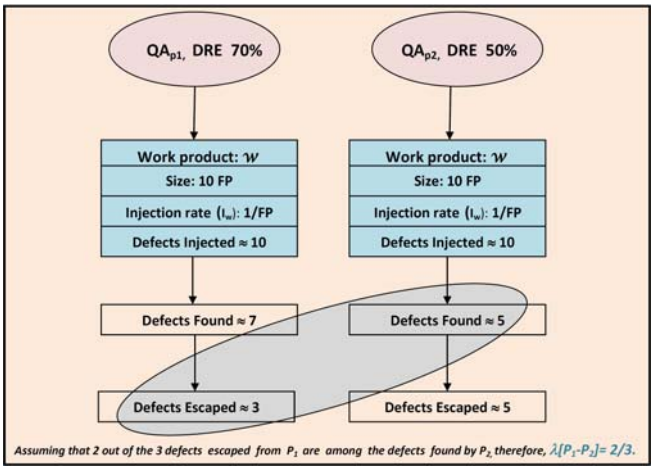

Figure 13: Lambda Variable Association

Based on **Figure 13**, the combination variable of applying the practice p_2 as a successor of p_1 is $\lambda = 2/3$. That is, QA practice QA_2 is able to uncover 2/3(two-thirds) of the escaped defects from practice QA_1.

By utilising the λ variable, the number of found defects after applying two QA practices sequentially is calculated by extending **Equation 2** as follows:

$$N_w^{Found} = \beta_{p_1} * eD_w * DRE_{p1} + \beta_{p_2} * eD_w * (1 - DRE_{p1}) * \lambda_{p1-p2}. \quad (9)$$

$p_1, p_2 \in P$

4.5 Return on investment

Defect detection and removal activities are considered to be an investment especially for profit-based software development organizations. This investment needs to be well-evaluated and studied to determine its positive and negative implications on the development process. The overall cost of any software development project is equal to the cost of the development activities and the cost of quality assurance practices implemented during the software life cycle.

Total development cost= Production development(COCOMO)+ Cost of quality

As our focus in this research is mainly on the second aspect of cost, which is the cost of software quality $CoSQ$, we will show how to evaluate the $CoSQ$ in a feasible way by using a well-known business measure, return on investment ROI. In our model, we express the ROI measure according to a value to cost ratio as shown in the following:

$$ROI = \frac{Value}{Cost} \tag{10}$$

Value : equals the saving in costs resulting from fixing defects early in their phases of origin.

Cost : equals the effort of both executing the QA practice and fixing defects found.

First of all, we recall the functions that we used in our model and constitute our model components. These functions are :

- Execution cost (Exc)
- Removal cost (Rc)
- Escaped cost (Esc)
- Saved cost (Sc)

In order to calculate the numerator for our ROI equation, (Value), we use the following expression :

$$\text{Value} = Sc - (Exc + Esc + Rc)$$

For each QA p applied to a piece of an artifact the estimated value is identified by subtracting the estimated saved cost from the other three aspects of execution effort: execution cost, defect removal cost and the escaped cost. The denominator in our ROI equation (Cost) can be defined as the sum of the three aspects of execution effort:

$$\text{Cost} = Exc + Esc + Rc.$$

Accordingly, substituting the two values in our ROI equation yields :

$$ROI = \frac{Sc - (Exc + Esc + Rc)}{Exc + Esc + Rc} \tag{11}$$

By generalizing the previous equation to the whole work product w, we can calculate the relative ROI of all QA plans applied to work product w by summing all variables as shown below:

$$ROI_w = \frac{\sum_{p \in P} Sc_p - (\sum_{p \in P} Exc_p + Rc_p + Es_p)}{\sum_{p \in P} Exc_p + Rc_p + Esc_p} \tag{12}$$

5: Related Work

Work related to our research includes similar models that work as a decision-based system to help project managers control the cost, schedule and quality of their software projects. One of the most well known cost and quality estimation model is the COnstructive QUALity MOdel (COQUALMO) [13] which is an extension to the COnstructive COst MOdel (COCOMO) [20]. It consists of two sub-models: 1, defect introduction and 2, defect removal models which in turn integrate into COCOMO to help determine the cost of defects removal using the three QA practices: *Automated analysis*, *Reviews* and *Testing*. However, In COQUALMO the effort to fix a defect introduced and removed by each of the three practices is not quantified directly by the model, it is calculated as a percentage of the total estimated effort by COCOMO model [6]. Also, along with the COCOMO, COQUALMO are calibrated based on data manipulation of many previous software projects which may incur difficulties for an organization to tailor any of those models to itself. Moreover, with COQUALMO there are only three applicable practices without taking into account the diversity of other techniques and the variations in their efficiency.

Capture recapture models is used in software engineering to predict the total number of defect in an artifact based on a sample taken from a population of defects [30][25]. However, the accuracy of defect estimations given by capture and recapture models is not stable and influenced by different factors like the type of QA practice [28], number of people involved [26] etc.

With regards to software QA practices and their defect detection and removal efficiency, Juristo et al [34] designed a classification schema for QA practices within software development life cycle upon a 25 years of testing experience. However, they concluded that there is a little knowledge related to QA practices and their applicable domain.

Gou et al [33] [16], analyzed data related to QA activities of historical projects and found that based on these data industrial baselines can be established to estimate defect removal and detection efficiency of current and future QA activities.

6: Conclusion

This paper is a part of ongoing research on a holistic model for software QA activities management. We proposed an approach that optimizes software investment assigned to QA processes by defining generic QA model whereby critical components within the software to be developed are given more consideration and priority .

The mechanism that our model relies on is that each phase deliverable is categorized into different work products according to predetermined risk rating levels introduced by software development organization in a way that high effective practices be applied to high risk rating level work products. We also proposed an adjustment to the defect containment matrix so as to make it able to determine the DRE for any risk rating level and accordingly the efficiency of each practice assigned to it. We presented set of mathematical equations that supports the theoretical model presented and to help determine the interaction and communication of our model aspects. Our QA model can be utilized as decision based QA system that helps software project manager make informative estimate on the consequences of daily based decisions regarding QA processes.

The aim behind this project research on which this paper is based is to come up with a holistic model to deal with software triple constraints schedule, cost and quality. There are many variables that were overlooked in this paper in order to make the proposed approach less complex.

Future research will include integrating the time consideration into the model so as to enable

triple constraints trade-off process. Ongoing contacts are being made with leading software development organizations to pilot the model for their software QA activities.

References

[1] Krasner.H, *Using the Cost of Quality Approach for Software*. CrossTalk. The Journal of Defense Software Engineering, pp11(11), 1998.

[2] Robert B., *Practical Software Metrics for Project Management and Process Improvement*, Englewood Cliffs, NJ: Prentice Hall, 1992

[3] Boehm. B and Huang.L. Value-based software engineering: A case study. Computer, pages 33-41, March 2003.

[4] Price Waterhouse, *Software Quality Standards: The Costs and Benefits*, A review for the Department of Trade and Industry. London: Price Waterhouse Management Consultants,1988.

[5] El Emam.K, *The ROI from Software Quality*, Boca Raton, Florida: Auerbach Publications, 2005.

[6] WANG,Q., GOU,L., JIANG,N., CHE,M., ZHANG,R., YANG,Y., LI,M. *An Empirical Study on Establishing Quantitative Management Model for Testing Process*, JCSP 2007, Lecture Notes on Computer Science 4470, 2007.

[7] Boehm.B, *Industrial software metrics top 10 list*, IEEE Software, September, pages 84-85, 1987.

[8] Jones.C,*Applied Software Measurement*, 3rd edition; McGraw-Hill, New York: 2008.

[9] Sower.V, Quarles.R, Cooper.S, *Cost of Quality:Distribution and Quality System Maturity: An Exploratory Study*, ASQ's 56th Annual Quality Congress Proceedings, ASQ, May, 2002.

[10] Fairley.R ,Willshire.M. , *Iterative Rework: The Good, the Bad, and the Ugly*. IEEE Computer 38 (9): 34-41, 2005

[11] ROBERTSON.S & ROBERTSON.R, *Mastering the Requirements Process*. Harlow, UK, 2006.

[12] LiGuo.H , Boehm.B, *How Much Software Quality Investment Is Enough: A Value-Based Approach*, IEEE SOFTWARE, 2006.

[13] Chulani.S, Boehm.B, *Modeling Software Defect Introduction Removal: COQUALMO (COnstructive QUALity MOdel)*, USC-CSE-99-510, The Center for software Engineering, University of Southern California, Los Angeles, CA, 1999.

[14] Votta.L, *Does Every Inspection Need a Meeting*, 1993.

[15] Florac, William, A., *Software Quality Measurement: A Framework for Counting Problems and Defects* , CMU/SEI-92-TR-22, Software Engineering Institute -Carnegie Mellon University, Pittsburgh, Pennsylvania, September 1992.

[16] Gou.L, Wang.Q, Yuan.J, Yang.Y, Li.M, Jiang.N, *Quantitative defects management in iterative development with BiDefect*, Software Process: Improvement and Practice, 14,4, pp.227-241, 2008.

[17] Chrissis. M.B, Konard. M, Shrum.S, *CMMI guidelines for process integration and product improvement*. Addison-Wesley,NJ, 2003.

[18] Boehm. B. and Basili, V. *Software Defect Reduction Top 10 List*, IEEE Computer, Vol. 34, No. 1, January 2001.

[19] Henderson.C, *Managing software defects: defect analysis and traceability*, ACM SIGSOFT Software Engineering Notes, 2008.

[20] Boehm B. W., *Software Engineering Economics*, Prentice-Hall, 1981.

[21] Chillarege.R, Bhandari.I.S, Chaar.J.K, Halliday.M.J, Moebus.D.S, Ray.B.K, and Wong.Y. Orthogonal Defect Classification-A Concept for In- Process Measurements, IEEE Transactions on Software Engineering, vol. 18, pp. 943-956. 1992.

[22] NASA., *NASA procedures and guidelines for mishap Reporting, Investigating and Record keeping*, Safety and Risk management Division, NASA Headquarters, USA, 2000.

[23] Humphrey.W.S, *A Discipline for Software Engineering*, Addison-Wesley Publishing Company,Massachusetts, 1995.

[24] Alshathry.O, Helge.J, Hussein.Z, Abdullah.A, *Quantitative Quality Assurance Approach*, niss, pp.405-408, 2009 International Conference on New Trends in Information and Service Science, 2009.

[25] Walia.G, Carver.J, *Evaluation of capture-recapture models for estimating the abundance of naturally-occurring defects*,Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement, pp.158-167, 2008.

[26] El Emam.K, Laitenberger.O, *Evaluating Capture-Recapture Models with Two Inspectors*, IEEE Trans. Software Eng., vol. 27, no. 9, pp. 851-864, 2001.

[27] Alshathry.O, janicke.H, *Optimizing Software Quality Assurance*, COMPSAC'10, The 34nd Annual IEEE International Computer Software and Applications Conference, 2010.

[28] ISODA.S, *A criticism on the capture-and-recapture method for software reliability assurance*. Journal of Systems and Software, 43, pp.3-10, 1998.

[29] L LAZIc, A KOLAcINAC, D AVDIC The Software Quality Economics Model for Software Project Optimization, World Scientific and Engineering Academy and Society (WSEAS) Stevens Point, Wisconsin, USA, 2009.

[30] Briand.L, El Emam.K, Freimut.B, *Comparison and Integration of Capture-Recapture Models and the Detection Profile Method*, Procs. Ninth International Conference on Software Reliability Engineering, Paderborn, Germany, pp. 32-41, 1998.

[31] Tsai.W. H., *Quality Cost Measurement Under Activity-Based Costing*, International Journal of Quality & Reliability Management, vol. 15, pp. 719-752, 1998.

[32] Capres.j, *Estimating Software Costs: bringing realism to estimatingý*, 2nd edition. McGraw-Hill, New York, 2007.

[33] Gou.L, Wang.Q, Yuan.J, Yang.Y, Li.M, Jiang.N, *Quantitatively managing defects for iterative projects: An industrial experience report in China*, Heidelberg, D-69121, Germany: Springer Verlag,pp.369-380, 2008.

[34] Juristo.N, Moreno.A, Vegas.S, *Reviewing 25 Years of Testing Technique Experiments*, Empirical Software Eng.,vol. 9, nos. 1Ű2, pp. 7Ű44, 2004.

[35] *Driving Quality Throughout the Software Delivery Lifecycle*, June 2007, Borland software corp. www.borland.com

[36] Wood et al, *Comparing and Combining Software Defect Detection Techniques: A Replicated Empirical Study*, Proceedings of the 6th European Software Engineering Conference, Zurich, Switzerland, pp. 262-277, 1997.

[37] Boehm.B,Valerdi.R, *The ROI of Systems Engineering: Some Quantitative Results*, eq-

uity,IEEE International Conference on Exploring Quantifiable IT Yields, pp.79-86, 2007.

[38] *Building a Better Bug Trap*, The economist , June19,2003,http://www.economist.com/science/tq/displayStory.cfm?Story_id=1841081 [accessed on] 16/oct/2009.

[39] Frost.A, Campo.M *Advancing Defect Containment to Quantitative Defect Management*, CrossTalk Ű The Journal of Defense Software Engineering 12(20), 24Ű28, 2007