http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 11 (1): 76-84, 2012
ISSN 1812-5638 / DOI: 10.3923/itj.2012.76.84

Enabling Continuous Quality Improvement with Quantitative Evaluation in Incremental Financial Software Development

[1]Bin Xu, [2]Meng Chen, [2]Cun Liu, [2]Juefeng Li, [2]Qiwei Zhu and [3]Aleksander J. Kavs
[1]College of Computer Science and Technology, Zhejiang University, 310027 Hangzhou, China
[2]State Street Technology (Zhejiang) 310030 Hangzhou, Peoples Republic of China
[3]State Street Corporation 02111 Boston, MA, USA

Abstract: While software play more critical roles in business, the quality of financial software systems become extremely important as the enterprises rely more on their software applications. Incremental software development enables the development team to yield the functionalities step by step so as to satisfy the changing business requirement and enable the high quality releases as well. However, efficient quality management is essential to make the best tradeoff between schedule, effort, cost and quality so as to reduce the potential risks in the business. In this study, the authors suggest a continuous quality improvement framework on the basis of quantitative quality evaluation in incremental financial software development. A set of evaluation, analysis and improvement approaches are described and designed. Related practice in a global IT corporation shows that the approaches have significant business value to support the best tradeoff making.

Key words: Quantitative quality management, incremental software development, quality evaluation, root cause analysis, quality management framework, statistics defect analysis

INTRODUCTION

While software applications are becoming increasingly large and complex due to the improvement of software development techniques, financial enterprises rely more than ever on some core software systems. Software quality is extremely important in financial systems with the goal to handle business transactions quickly, accurately and reliably, to protect the business privacy and successfully deal with kinds of exceptions. Though the delivery date, budget and resource effort can be negotiated most of the time, software quality is a very important criterion for the customer to accept the product and must not be jeopardized. Bad quality software applications frequently results in revenue loss, hurts the user confidence, unable to provide competitive advantage to business and creates additional workload (Original Software, 2010).

Incremental development model (Mills *et al*., 1987) is adopted to handle requirements gradually and develop the system in steps that accumulate the functionalities. It allows partial utilization of product, shortens the development time and helps ease the traumatic effect and risk of introducing completely new system all at once (Sommerville, 2001). While it takes advantage of flexibility in resource utilization (Ruhe, 2005), incremental software development model also brings forth some problems. First because each increment is developed and integrated into the application consequently, much effort is required if the previous increments were poorly designed, hard to be understood and difficult to be enhanced (Xu, 2005). Defects slip from former increments may be mixed and amplified which may introduce new problems to the following increments and thus increase the effort to fix them (Xu, 2010). Second, the turnover of human resource requires knowledge sharing effort and further increases the effort in subsequent increments development across different increments (Xu and Pan, 2006). Third, more requirement changes are expected in incremental development and if not properly managed or controlled, development will be definitely led into chaos (Xu, 2005; Xu *et al*., 2004). Quality management (Rose, 2005) after each increment or even during one increment is necessary to solve these problems.

There are four main components in quality management: Planning, control, assurance and improvement (Rose, 2005). It is not only a principle that ensures the high quality in software products and services but also the meaning to control processes. Quality management therefore uses quality assurance in software product and controls processes as well as products to achieve more consistent quality. Quality management is important to companies to have higher product performance, increased revenues,

Corresponding Author: Bin Xu, College of Computer Science and Technology, Zhejiang University, Hangzhou Zhejiang, 310027, China Tel: +86 13666636919

better customer satisfaction and smaller waste (Ahire, 1997). 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.

The objective of this study is to clarify the special quality related problems in the incremental financial software development and then enable the management to understand and oversee the quality status of the increments so as to make the best tradeoff in quality management and avoid the insufficient testing and optimize the quality assurance tasks.

MOTIVATIONS

There are not many works that focus on quality management in incremental software development. Hewett (2011) suggested mining software defect data to support software testing management. A method is proposed on how to order requirement in incremental development to improve quality management (Wohlin, 1994) and Unified Process is chosen as the embedded framework to perform the quality management (Jacobson *et al*., 1999; Norbjerg, 2002; Kroll and Kruchten, 2003; Shuja and Krebs, 2007).

Unfortunately, quality is often been ignored when the project is short of schedule or budget. The requirement on a specific release date often overrules quality objectives, for example reliability. A software product may be delivered on time but not thoroughly tested or verified before release due to delays. In other words, testing effort is cut off because of the tough schedule which results in poor quality software released to client (Original Software, 2010). In order to ensure the high quality in the incremental software development, quantitative quality management is essential to measure the quality and the risk of low quality. However, the quality characteristics defined in the ISO/IEC (2001) is not practical to be measured directly, lower-abstraction attributes of the product should be accessed from ISO/IEC (1998).

With the years of incremental financial software development, the authors found that the poor quality increment resulted endless quality risks, brought more rework effort and continually hurt the team confidence in the following increments before the related defects in the former increments had been finally fixed. A poor-quality-delivery of an increment brings forth lots problems because:

- Defects inside former increments may be amplified in the following increments
- After-increment defect removal requires much more rework effort and impact a serial of tasks across requirement, design, coding and testing phases
- Worse quality status gives critical pressure on the schedule management of consequent increments and would make the following quality status even worse

In this study, the authors argue that quantitative management techniques should be adopted in software quality management and suggest a series of practical solutions to evaluate, analyze and improve the software quality in incremental financial software development.

QUANTITATIVE SOFTWARE QUALITY EVALUATION

Quantitative quality management framework: Quantitative quality management framework is a dynamic framework which could be improved during the incremental software development practice. It starts at evaluation phase, within which the quality and the quality management will be evaluated with some defined rules and criteria. After the evaluation, some further analysis could be done on increments with potential issues. The rules, criteria and the factors could be calibrated and improved to reduce the total effort in quality management. The whole framework generates a loop which enables the continuous quality improvement (Fig. 1). All these four steps could be overlapped in practice, for example, the evaluation can be made during the practice and the improvement can be suggested after the practice has started.

Indicate the possible quality issues after increment: Bug density can be a good metric to measure the quality of an increment by comparing the subsequent increments in an incremental software development project. Generally, a

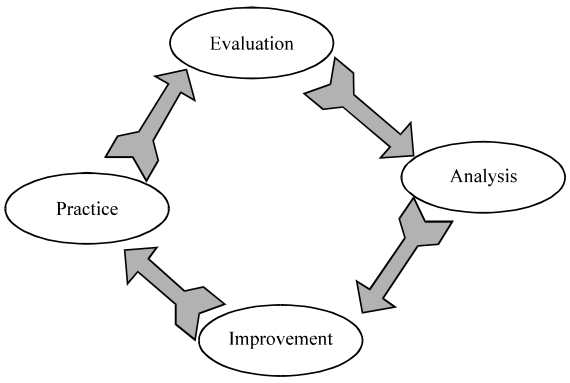

Fig. 1: Quantitative quality management framework

lower bug density means High quality while a higher one means Low quality which will be the subject of later analysis.

In this study, bug density is viewed in another view and used to indicate the possible quality issues after increments. Bug density will be estimated according to some historic project from the benchmark repository. The estimated bug density will be calibrated with the average value during the progress of the incremental software development. We assume that there are some problems in those extremely High or extremely Low quality increments. For example, the testing team had much more time to test the increments and found more defects than the average for some Low quality increments. For the High quality increments, maybe the testing team didn't have sufficient time to test the increments and found far too little defects than the average.

In this study, we focus on the extremely High quality increments. Several criteria are set to indicate the possible quality issues:

- Scale is large but complexity is not lower. Line of source code (ESLOC) is used to evaluate the scale and scorecard is used to measure the complexity
- Test duration is short
- Product management is highly efficient and does an excellent job for resource allocation and schedule planning

Root cause analysis: Causal analysis and resolution is a CMMI process at level 5. It contains two Specific Goals which is split into five Specific Practices (Buglione and Abran, 2006).

Root cause analysis enables the team to find out the weakness of the development so that the team could improve the quality of the increments in the future by avoiding to the similar cause.

An analytical tool from Total Quality Management (TQM) (Deming, 1986; Juran and Gryna, 1993; Crosby, 1979) named the Fishbone diagram (Yu, 1998) is useful for detecting the root causes of a software defect/problem and for classifying and prioritizing issues.

For each fault fixed, quality assurance engineers and developers choose a fault type and one or more root causes. Based on the fault data collected from developers and quality assurance engineers, most actual faults could be covered. Fishbone diagram for defect cause analysis is suggested by Buglione and Abran (2006).

Statistics analysis on defects from root cause perspective: Root causes described in the fishbone diagram can be considered as the multiple-dimensions, for example, SDLC original causes were used as the top categories in this research. Some of such causes can be Requirement misunderstanding, Requirement unclear, Requirement faulty, Detail design failure and etc.

Statistics analysis: Defect report is generated after the root cause analysis. In order to enable the decision making in the quality improvement, the price in bug fixing is estimated in the unit of working hours according to manual experience.

QUANTITATIVE QUALITY IMPROVEMENT

Solutions for quality improvement: There are many solutions for quality improvement while these solutions requires different invest of budget or human resource. For example, financial budget is required to purchase a better testing tool and some training effort is also needed for the test engineers to perform this new testing tool. Such financial budget investment and the training effort are fixed cost which is not dependent on the number of defects.

The improvement solutions include the follows, just to name some of them.

To reduce defect of Coding:

- Enhance development test
- Involve code review
- New technology training

To reduce defect of Requirement Misunderstood/ Unclear:

- Requirement review meeting
- Requirement question list

To reduce defect of Faulty Requirement:

- Involve more clients interactive
- Enhance business analysts role

The solutions for the quality improvement are not limited to solve only one defect category. For example, the software review may reduce the defects in the coding and benefit the requirement understanding. In such case, the related benefit and cost is different for each category.

For each root cause category, there could be several improvement solutions with different benefit and cost. Typically the benefit is measured in the unit of the Defect Removal Efficiency (DRE) and the cost refers to both the duration and effort to avoid such defects through review or similar techniques. The values of benefit and cost

should be benchmarked and calibrated for individual IT projects according to its local historic experience.

RELATED DEFINITIONS IN THE IMPROVEMENT

Generally, Pareto analysis will be performed to find out the possible defect category to be improved. For each category to be improved, a suitable working plan should be determined according to the number and price of the defects and related benefit and cost of the solutions.

Definition 1

Summarized report of root cause analysis: Summarized report of root cause analysis can be defined as a matrix SRCA :: = <RC, N, P, BV> where RC is the top category of the root causes, N is the number of the related defects, P is the estimated bug fix effort and BV is the business value after fixing the defects.

Typically, high quality of software refers to low defect density in the software. Here in this research, the business value is considered as the weight of quality when comparing defects across different categories. In order to improve the quality continuously, the project team should clarify the status of defects and remove the defects efficiently, that is obtain more business value with less bug fix effort.

Hot defect categories refer to the defect categories which should be reduced in the increments so as to enable the high software quality. Business value and bug fix effort will be tradeoff to obtain high quality in the increment. Most of the time, project team needs to enlarge the business value with limited budget for the bug fix effort.

Definition 2

Improvement solution: IS :: = <ID, Name, FixedCost>, where FixedCost is the cost which is independent from the number of defects or the time of implementation. The FixedCost is important for the project managers to make the decision to try a new testing tool when the Fixed Cost is affordable and the related execution cost is small when compared with some other solution.

Definition 3

Capacity of improvement solution: The capacity can be defined for the quality improvement towards different defect category. CIS :: = <RC, IS, DRE, Cost, N, SEffort> where RC is the root cause category, IS is the improvement solution, DRE is the possible defect removal efficiency to remove the defect category RC with the solution IS and Cost is the execution effort for each implementation, for example, the additional requirement review effort. The execution cost varies from different defect category and different improvement solution. N and SEffort refer to the number and summarized effort for the defect in the category of RC which detected by the solution IS. These two items are prepared for the calibration of the capacity.

The capacity will be impacted by the team capacity, the technique complexity and the project environment. Therefore, capacity of improvement solution will be calibrated and updated with more project experience so as to be shared with multiple projects.

Definition 4

Quality assurance task organization: In some case, one improvement solution may exposure the defects in multiple categories. For example, the functional testing may help to find out the requirement, design and coding related defect. The organization of quality assurance task is defined as QTO :: = <Task, IS, QTI> where Task is the task category for quality assurance work, IS is the improvement solution and QTI is queue of <RC, Ratio> where RC is the defect in root cause category and Ratio is the percentage of effort.

Definition 5

Quality assurance task execution: In the execution of each quality assurance task, defects in different categories may be detected. The execution of quality assurance is defined as QTE :: = <Id, Task, IS, Effort, DefectFound> where Id is the task identification of qte. Task is the task category for quality assurance work, IS is the improvement solution, Effort is the effort to finish task qte and DefectFound is queue of <RC, N> where RC is the defect in root cause category, N is the number of defects found in the execution of qte.

Definition 6

Quality improvement system: The entire quality improvement system can be defined as QIS:: = < SRCA, IS, CIS, $BV_{expected}$, Budget> where SRCA is the summarized report of root cause analysis, IS is the set of improvement solutions, $BV_{expected}$ is the expected business value in the improvement and Budget is the reserved budget for the bug fixing.

Algorithms for the quality improvement: When there is no sufficient time for the bug fixing, the defects couldn't be fixed within one increment. However, the quick bug fix will benefit the product quality and reduce the development of future increments. In this study, to

obtain the maximal business value within the bug fix budget is used as the criteria. Algorithm 1 is used to identify the defect categories which should be fixed for the better quality and the defects in other categories may be considered to be fixed in the next increment.

Algorithm 1: Identifying hot defect categories

Input: Summarized Report of Root Cause Analysis SRCA and reserved budget Budget.
Output: hot defect categories HDC, expected business value $BV_{expected}$.
1 sort SRCA in the descending order of BV/P
2 BVexpected = 0, cost = 0, HDC = Φ
3 FOR (each srca in SRCA) {
4 IF (cost > = Budget) break;
5 IF (cost+srca.N* srca. p <= Budget) {
6 cost= cost+srca.N* srca.P;
7 $BV_{expected}$ = $BV_{expected}$+srca.N * srca.BV;
8 add srca.RC into HDC
9}
10}

Algorithm 1 walks through the summarized report of root cause analysis and identify the hot defect categories to be reduced in the increment. The defect category is queued in the descending order of business value/bug fix effort. In such way, the quality improvement could be achieved in the most efficient way which has the largest return on investment.

The bug fixing effort is limited by the reserved budget and the expected business value is estimated as the summary of the number and the business value per defect. Of course, the bug fixing effort contains the re-testing and re-opening effort.

Algorithm 2: Calibrating of the capacity of improvement solution

Input: Quality assurance task organization QTO, Quality assurance task execution QTE, Summarized Report of Root Cause Analysis SRCA, defect remained DR and Capacity of Improvement Solution CIS.
Output: calibrated Capacity of Improvement Solution CIS.
1 FOR (each cis in CIS)
2 cis.N = 0, cis.SEffort = 0;
3 FOR(each dr in DR)
4 dr.defect_found = 0;
5 FOR(each srca in SRCA)
6 srca.N = 0;
7 FOR (each qte in QTE) {
8 Fetch qto from QTO where qto.task = qte.task;
9 FOR (each qti in qto.QTI) {
10 Fetch cis from CIS where cis.RC = qti.RC and cis.IS = qte.IS;
11 cis.SEffort = cis.SEffort+qti.Ratio*qto.Effort;
12}
13 FOR (each defectfound in qte. DefectFound) {
14 Fetch cis from CIS where cis.RC = defectfound.RC and cis.IS = qte.IS;
15 cis.N = cis.N+defectfound.N;
16 Fetch dr from DR where dr.RC= defectfound.RC;
17 dr.defect_found = dr.defect_found+defectfound.N;
18}
19}
20 FOR (each cis in CIS) {
21 Fetch dr in DR where dr.RC=cis.RC;
22 cis.DRE = cis.N /(cis.N+dr. defect_remained);
23 cis.Cost = cis.SEffort / cis.N;
24}

Algorithm 2 initializes the capacity of improvement solution CIS, defect remained DR and root cause analysis report SRCA at first, then update the summarized defect into CIS, SRCA and DR as well. The effort for the defect detecting is fetched from the quality assurance task execution QTE. Finally the DRE and Cost are calibrated in CIS.

Here, DRE is calculated as ΣDefect_found/ (ΣDefect_found+Defect remained) for each defect category. The number of defect remained is the feedback from business side after the increment.

For each phase, there are a serial of development and Quality Assurance tasks. From the perspective of quality assurance, development tasks introduce the defects and Quality Assurance tasks help to explore the defects.

After the benchmark has been prepared, the defects numbers of a new increment can be estimated according to some predict approach. In Algorithm 3, we suggest an approach to schedule the improvement solutions to the coming increment. E-SRCA is a matrix $E\text{-}SRCA_{rc, ph}$ which is estimated according to the historic data, where rc is the defect category identified by root cause, ph is the phase number stated from 1 to 4, refer to Inception, Elaboration, Construction and Transition. E-DR is a vector $E\text{-}DR_{rc,ph}$ which is estimated according to the historic data as well where rc and ph have same meaning as in E-SRCA but $E\text{-}DR_{rc,ph}$ refers to the number of defects in the category of rc will be remained after phase ph. The structure of scheduled improved solutions is defined as a vector SIS_{ph} and each item refers to a queue of improvement solutions <IS>.

Algorithm 3: Scheduling improvement solutions

Input: Quality assurance task organization QTO, Quality assurance task execution QTE, estimated Summarized Report of Root Cause Analysis E-SRCA and defect remained E-DR and calibrated Capacity of Improvement Solution CIS.
Output: Scheduled improvement solutions SIS, estimated effort E-Effort.
1 E-Effort = 0;
2 SORT CIS in the descending order of CIS.COST;
3 FOR (each ph in 1..4) {
4 SIS_{ph} .initial();
5 qtr = $E\text{-}SRCA_{ph}$;
6 While (qtr.totalnumber() > $E\text{-}DR_{ph}$.totalnumber())
7 FOR(each is in CIS.IS)
8 IF ($E\text{-}SRCA_{is.RC,ph}$>0) {
9 SIS_{ph} .add(is);
10 FOR(each rc in CIS.RC where CIS.IS=is) {
11 E-Effort = E-Effort + CIS.COST * qtr_{rc} *(1- CIS.DRE);
12 qtr_{rc} = qtr_{rc} *(1- CIS.DRE);
13}
14 ELSE continue;
15}

Algorithm 3 is briefly demonstrated with some vector access, such as the line 5. The member method total number () is used to calculate all the defects number of a

vector. For each phase, the total removed defects number should be no less than the predefined number in E-DR. SIS_{ph}. Initial () at line 4 is used to initial the vector as a empty queue. SIS_{ph}. add (is) at line 9 is used to append the improvement solution is to the vector SIS_{ph}. The scheduling improvement solution in SIS together with the E-Effort will be output for the quality assurance task scheduling.

CASE STUDY IN GLOBAL IT CORP

We applied our approach in a financial software company which provides advanced IT solutions to a top financial organization.

As usual, we collected bug density increment by increment as shown in Fig. 2. We found that the increment R003's bug density was fairly lower than the average bug density, even lower than the average minus one sigma. Therefore, we checked the criteria:

- The scale is LARGE but the complexity is NOT lower. The features in the increment R003 were more complicated than the average based on the scorecard from development and testing teams
- Test duration is very short comparing with other increments
- As a surprise there was no sufficient product management for R003

As a result, the increment R003 was indicated to have quality issues: Test cycle was not sufficient based on the stable testing resources across the multi-increments and the short testing duration.

S-Curve which has been introduced in Section 2.3 was used to validate our assumption. S-Curve was generated as in Fig. 3 to demonstrate the expected and exposed defects number. There was a big gap between expected and exposed defects number for the increment R003: Only half of the expected defects had been found. This was an important evidence to support our assumption.

We verified this assumption with the project team and got the confirmation that the testing for the increment R003 was insufficient. Due to the tight schedule of the increment R003, testing duration was cut short which caused more after-release defects. It was really a good example of improper cutting testing efforts under tight schedule as many defects were found on clients' site.

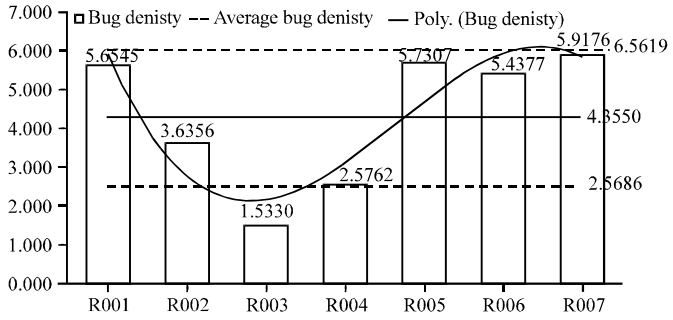

Fig. 2: Bug density analysis on the increment (released to client)

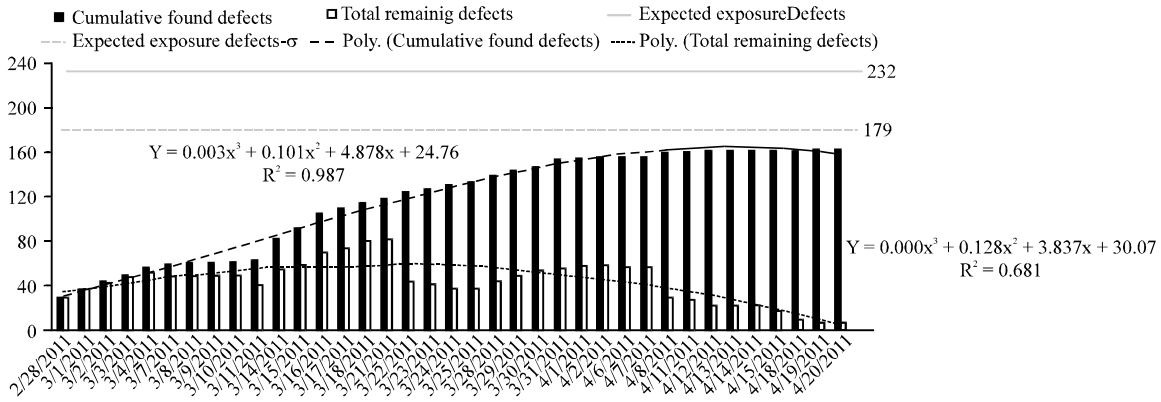

Fig. 3: S-Curve analysis on the increment R003

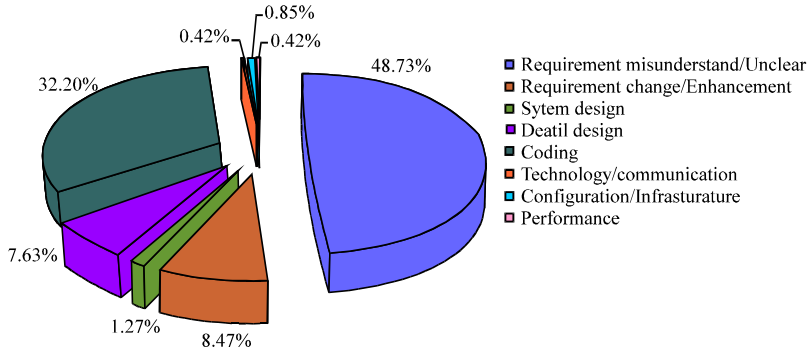

Fig. 4: Defects distribution by root cause

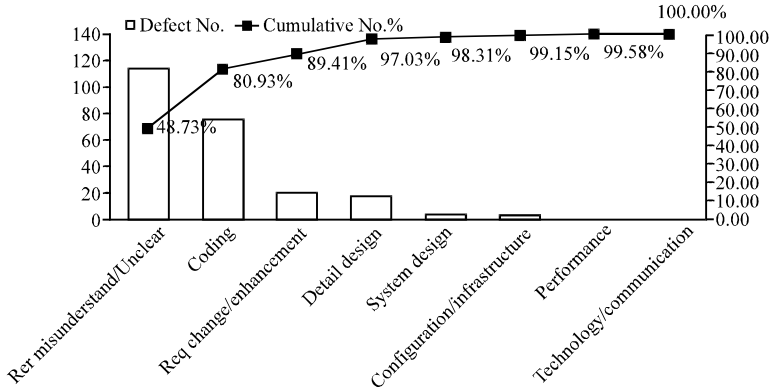

Fig. 5: Pareto chart for the root cause report

After we had indicated the increment with high quality issue (though it looked like an extremely High quality increment), root cause analysis was used to clarify the defects in the distribution of root cause, as is shown in Fig. 4.

Pareto chart was used to identify the most important root causes that must be eliminated or diminished to improve the quality to an acceptable level. As shown in Fig. 5, it is clear that Requirement Misunderstand/Unclear and Coding were the main two causes for the increment R003.

According to our experience, the possible improvement solutions to reduce the requirement misunderstand/unclear related defects are enhancing the requirement review and establish requirement issue list. We suggested to the project team to enhance the requirement review in the following increments and host additional requirement review meetings for the previous increments to reduce the impact of the quality issue from R003. The decision making for the quality improvement should be made according to the feature of the project and the capability of the project team.

DISCUSSION

Many software quality metrics have been developed to verify the traceability (Singh and Saha, 2010), 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; Vinayagasundaram and Srivatsa, 2007; Bedi and Gaur, 2007). However, a single metrics means little sense and additional information or metrics are required to determine the status of quality. As demonstrated in Fig. 2, the bug density of R003 is far below the average of the other increments. Typically it will be regarded as the high quality of the increment. In this study, we found that the bottom boundary of the bug density is 2.5686 and wondered if there was some quality issue in R003. With the survey from the team, we found that the test duration was too short and the product management was not sufficient. Therefore, we concerned that the testing on R003 is insufficient. Late, some feedback from production testing verified that there were much more defects remained than other increments. The S-curve in Fig. 3 showed that the speed (efficiency) of the

testing was much higher than other increments. Both defect distribution diagram in Fig. 4 or Pareto chart in Fig. 5 showed that the requirement misunderstand/unclear and coding accounted for too much ratio of the total defects. This could be considered as another low quality evidence of this increment.

In this study, we made an assumption that the solution adopted in the testing phase would remove the defects in fixed ratio regardless of the priority or order of the adoption. Suppose that there are two improvement solutions, QA_1 and QA_2, the defect removal efficiency of QA_1 is DRE_1 and QA_2 is DRE_2. The defect removal efficiency of testing queue $<QA_1, QA_2>$ is same as that testing queue $<QA_2, QA_1>$ which equals to $100\%-(100\%-DRE_1)*(100\%-DRE_2)$.

Assume DRE_1 of QA_1 is 80% and DRE_2 of QA_2 is 70% and there are 100 original defects. If QA_1 is done at first, then it could explore 80 defects and QA_2 could only explore 20*70% = 14 defects. If QA_1 is done after QA_2 then QA_2 could explore 70 defects and QA_1 could only explore 30*80% = 24 defects. There are 6 remain defects for both scenarios. However, the improvement solutions implemented in different orders result in different evaluation on the improvement solutions and will definitely yield different optimization results.

Pareto technique has been suggested to value the conditional dependence (Barro, 2009), optimize the service composition and to enhance the evolutionary algorithm (Zhihuan *et al*., 2010). In this study, we used Pareto chart to identify the most significant defect category as in Fig. 5. However, the solutions could not be limited to only those most significant categories if we consider the related implementation cost. Currently we don't have enough evidence to validate this suggestion but we will continue to gather the project data and it will be further studied with plenty of project data in our future work.

The framework and approaches suggested in this study can also be used in some other quality-critical incremental software development. The historic project data should be prepared and be used to calibrate the benchmark. This will be the future work of the authors.

CONCLUSION

In this study, we suggested a quantitative quality management framework with a series of analysis and evaluation approaches. It has been used in a real project and the experience showed its value in identifying the quality issue which may not have been easily found manually with some traditional approaches. In such way, the poor quality increment may be reworked in time so as to avoid the possible problems, such as the amplified defects number, more rework work due to late defect identifying and fixing and low team confidence because of low quality status.

ACKNOWLEDGMENTS

This work is part of "Global Collaborative Software Development" research project which is an attempt to improve the dual-shore software development with integrated best practice, software engineering technology and project management methodology. The research project is funded by State Street Corporation, USA. The project is collaboration between Zhejiang University, China and State Street Corporation, USA.

REFERENCES

Ahire, S.L., 1997. Management science-total quality management interfaces: An integrative framework. Interfaces, 27: 91-105.

Barro, D., 2009. Conditional dependence of trivariate generalized pareto distributions. Asian J. Math. Stat., 2: 20-32.

Bedi, P. and V. Gaur, 2007. Trust based quantification of quality in multi-agent systems. Inform. Technol. J., 6: 414-423.

Boroni, G. and A. Clausse, 2011. Object-oriented programming strategies for numerical solvers applied to continuous simulation. J. Applied Sci., 11: 2723-2733.

Buglione, L. and A. Abran, 2006. Introducing root-cause analysis and orthogonal defect classification at lower CMMI maturity levels. Proceedings of the International Conference on Software Process and Product Measurement, November 6-8, Cadiz, Spain, pp: 29-40.

Changchien, S.W., J.J. Shen and T.Y. Lin, 2002. A preliminary correctness evaluation model of object-oriented software based on UML. J. Applied Sci., 2: 356-365.

Crosby, P.B., 1979. Quality Is Free: The Art of Making Quality Certain. McGraw Hill, New York, USA., ISBN-10: 0-451-62585-4, Pages: 270.

Deming, W.E., 1986. Out of the Crisis, 1st Edn., MIT Press, Cambridge, ISBN: 0911379010. pp: 98.

Hewett, R., 2011. Mining software defect data to support software testing management. Applied Intell., 34: 245-257.

ISO/IEC, 1998. International standard, information technology: Software product evaluation, Part 5: Process for evaluators. International Standards Organization.

ISO/IEC, 2001. Software engineering: Product quality, Part 1: Quality model. International Standards Organization.

Jacobson, I., G. Booch and J. Rambaugh, 1999. The Unified Software Development Process. 1st Edn., Addison-Wesley Longman, MA., USA., ISBN-10: 0-201-57169, Pages: 512.

Juran, J.M. and F.M. Gyrna, 1993. Quality Planning and Analysis: From Product Development through Use. 2nd Edn., McGraw-Hill, New York, USA., ISBN-10: 0-07-033178-2, Pages: 629..

Kroll, P. and P. Kruchten, 2003. The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP, Addison-Wesley, MA., USA., ISBN-10: 0321166094, Pages: 416.

Mills, H.D., M. Dyer and R.C. Linger, 1987. Cleanroom software engineering. IEEE Software Mag., 4: 19-25.

Norbjerg, J., 2002. Managing incremental development: Combining flexibility and control. Proceedings of the European Conference on Information Systems, June 6-8, Gdansk, Pol and, pp: 229-239.

Original Software, 2010. Application quality management survey result. http://www.origsoft.com/ products/ qualify/docs/aqm_survey_results.pdf

Pan, L. and B. Xu, 2010. Towards collaborative master student talent development with E-CARGO model. Inform. Technol. J., 9: 1031-1037.

Parthasarathy, S. and N. Anbazhagan, 2006. Analyzing the software quality metrics for object oriented technology. Inform. Technol. J., 5: 1053-1057.

Rose, K.H., 2005. Project Quality Management: Why, What and How. J. Ross Publishing, Fort Lauderdale, Florida, USA.

Ruhe, G., 2005. Software Release Planning. In: Hand book of Software Engineering and Knowledge Engineering, Chang, S.K. (Ed.). UK pp: 365 394.

Shuja, A. and J. Krebs, 2007. IBM Rational Unified Process Reference and Certification Guide: Solution Designer, IBM Press, UK.

Singh, Y. and A. Saha, 2010. Predicting testability of eclipse: A case study. J. Software Eng., 4: 122-136.

Sommerville, I., 2001. Software Engineering. 6th Edn, Addison-Wesley, USA.

Vinayagasundaram, B. and S.K. Srivatsa, 2007. Software quality in artificial intelligence system. Inform. Technol. J., 6: 835-842.

Wohlin, C., 1994. Managing Software Quality through Incremental Development and Certification. In: Building Quality into Software, Ross, M., C.A. Brebbia, G. Staples and J. Stapleton, (Ed.). Computational Mechanics Publications, Southampton, United Kingdom, pp: 187-202.

Xu, B. and X.P. Pan, 2006. Optimizing dual-shore SQA resource and activities in offshore outsourced software projects. Proceedings of the 19th Annual Canadian Conference on Electrical and Computer Engineering, May 2006, Ottawa, Ont., pp: 2405-2409.

Xu, B., 2005. Extreme programming for distributed legacy system reengineering. Proc. 29th Ann. IEEE Int. Comput. Software Applic. Conf., 2: 160-165.

Xu, B., 2010. Cost efficient software review in an E-business Software Development Project. Proceedings of the 2010 International Conference on E-Business and E-Government, May 7-9, Guangzhou, pp: 2680-2683.

Xu, B., X.H. Yang, Z.J. He and S.R. Maddineni, 2004. Achieving high quality in outsourcing reengineering projects throughout extreme programming. Proceedings of the 2004 IEEE International Conference on Systems, Man and Cybernetics, Oct. 10-13, IEEE, pp: 2131-2136.

Yu, W.D., 1998. Software fault prevention approach in coding and root cause analysis. Bell Labs Tech. J., 3: 3-21.

Zhihuan, L., L. Yinhong and D. Xianzhong, 2010. Improved strength pareto evolutionary algorithm with local search strategies for optimal reactive power flow. Inform. Technol. J., 9: 749-757.