Research Journal of Applied Sciences, Engineering and Technology 7(1): 191-197, 2014
DOI:10.19026/rjaset.7.239
ISSN: 2040-7459; e-ISSN: 2040-7467
© 2014 Maxwell Scientific Publication Corp.
Submitted: June 13, 2013 Accepted: June 28, 2013 Published: January 01, 2014

Research Article

A New Dynamic Model for Software Testing Quality

[1]Akram Moustafa, [1]Abdusamad Al-Marghirani, [2]Mohammed M. Al-Shomrani and
[3]Ahmad A. Al-Rababah
[1]Northern Border University, KSA
[2]King-Abdulaziz University, KSA
[3]Hail University, KSA

Abstract: Research and study of Software Quality has Traditionally Focused on the Overall Product Quality Rather than on the sub-phase's milestones. There are many attempts over software testing as a standalone development phase which introduced in the literature; these efforts lacked the dynamic nature which has a diverse effect on the maintenance phase. This study will present the current software testing models; their challenges and at the end it will present our new dynamic model for applying quality assurance requirements over the sub-phases of the testing model. Quality properties may include as performance, efficiency, reliability, etc., new model will present for Improving the effectiveness and efficiency of the testing process through applying the quality requirements, designing high quality products, producing software with high Cost optimization, satisfying the product stakeholders.

Keywords: Dynamic model, software development life cycle, software quality, software testing, software verification and validation

INTRODUCTION

IT Professionals have various kinds of opinions on many software development principles, but mostly IT Professionals agree with one thing above all, Whatever software that is delivered into the market it must be accurate and reliable and successful software recognition for the long time is based on effective testing which meets the goal (Pressman, 2005; Dan and Russ, 2008; Timothy and Robert, 2005; Thayer and Christiansen, 2005). In a recently survey of software development, it is found top-of-mind issue is software testing and quality assurance (Software Engineering Institute, 2008; Pine *et al*., 2008; Aranda *et al*., 2007; Aggarwal and Yogesh, 2005). Testing is not a quality assurance; it and badly designed will be a bad product. However, software testing is one of the core technical activities that can be used to improve the quality of the software (Pine *et al*., 2008; Roger, 2005; Gottesdiener, 2005). Testing is a collection of techniques which is used in measure and improves software quality.

Testing gives us a broader category of software management practices which are known as quality assurance as well as other testing related things are defect tracking, design and code inspections (Pekka *et al*., 2007; Lewis, 2008). Before and After test execution test activities exist such as planning and control, choosing test conditions, designing test cases and checking results, evaluating completion criteria, reporting on the testing process and system under test and finalizing after the test phase has been completed (Pine *et al*., 2008; David and Hossein, 2005; James, 2009; Chandrasehakhar, 2005).

All software development models should include a testing phase as a mandatory phase through which the product may meet the end users requirements. Software testing can be stated as the process of validating and verifying that software meets the implementation requirements and guided its design and development (Karl, 2006; Pressman and Ince, 2007). The software testing process takes all individual units for the initial testing that may followed by the integration of these tested units in one module to be then tested and integration with other modules to give us the tested subsystem and finally these tested subsystems are compound to produce the final overall software tested system (Pressman and Ince, 2007; Schulmeyer, 2007). Software testing depends on the testing methodology and can be implemented at any time in the development process (Verification and Validation). Software testing in itself cannot ensure the quality of software. On its own, all testing can do is providing a tested software over the test cases and used samples. This testing challenge will be taken into consideration in this underlying study (David and Hossein, 2005; Weinberg, 2008; Schulmeyer, 2007).

Corresponding Author: Akram Moustafa, Northern Border University, KSA
This work is licensed under a Creative Commons Attribution 4.0 International License (URL: http://creativecommons.org/licenses/by/4.0/).

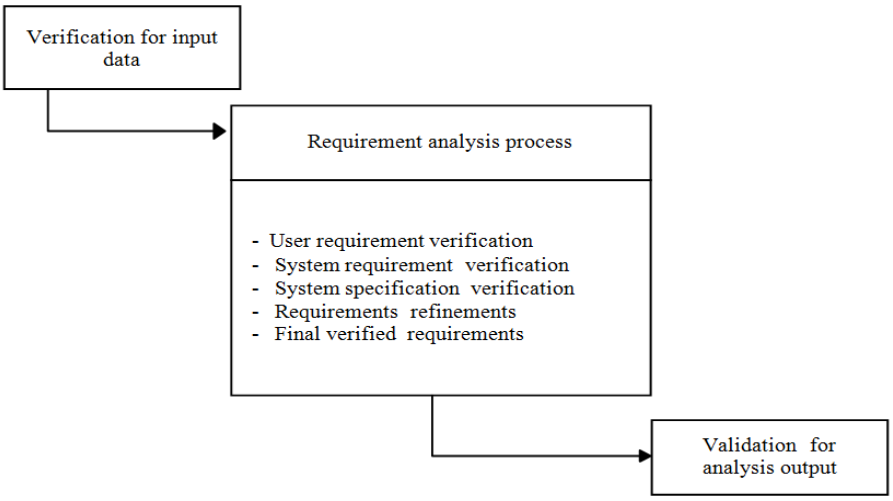

Fig. 1: Analysis based on verification and validation

SOFTWARE SYSTEMS MANAGEMENT

Software systems are more frequent part of the life, from consumer products to business products. Most people have had an experience with the software that did not work as expected. Software not working correctly can be creating many problems. Then it cans loss lot of credibility as well as loss of money and time. Human can make mistakes in software which produce a fault in the code, in software or in a system or in a documentation part. If a defect code is executed then the system will fail. These types of defects in software systems or documents may result in failures, but not all defects do so. Also mistakes may be faced, because of the time limits, complex type of code, changing the technologies and many system interactions.

Software management is a set of practices that attempt to achieve the following advantages: Deliver the software products with the expected functionality and quality, deliver the software on the expected time; deliver the software within the expected cost and meet expected levels of service during the software uses.

Verification and Validation (V and V): In the aspect of quality assurance in software development, documenting and designing the software. Developers can follow corporate standard processes but it does not mean quality assurance is responsible for that product. This is not a responsibility of the testing team because testing team can't improve quality; they can only measure it, although designing test before coding start will improve quality because user can think and use the information about their designs and during coding and debugging. Verification normally involves reconsiders and meetings to evaluate plan, code, obligations and specifications. This can be done new with checklists, matters registers, walkthroughs and inspection meetings.

Validation typically engages genuine testing and takes location after verifications are completed, (Fig. 1).

Software testing has three main features: Verification, Validation, Defect finding. Verification process verifies that the software is meeting all the technical specifications.

A specification is the part of the description in terms of a measurable output value given a specific input value under specific pre-conditions. Validation process confirms that the software meets the business requirements. Defect finding is just like variance between the expected and actual results.

MATERIALS AND METHODS

Test plan: An incomplete test plan will give the failure results which checks how the application works on different hardware and operating systems. There may be more than a few possible system combinations require that needs to be tested. System Testing tests all the modules and integrated components of the complete application. System test which may require involvement of other systems although should be minimized as much as possible to reduce the risk problem, (Fig. 2). This system testing interacts with other parts of the system comes in integration testing.

System testing requires many test runs because it involve as a necessary feature by feature validation of behavior using a wide range of both normal and erroneous test inputs and data. The test plan is critical because it contains descriptions of the test cares, the sequence in which the tests must be executed and the needed to be collected in each run.

Proposed U-lifecycle model: This model represents the software development process which may be considered as an extension of the process model.

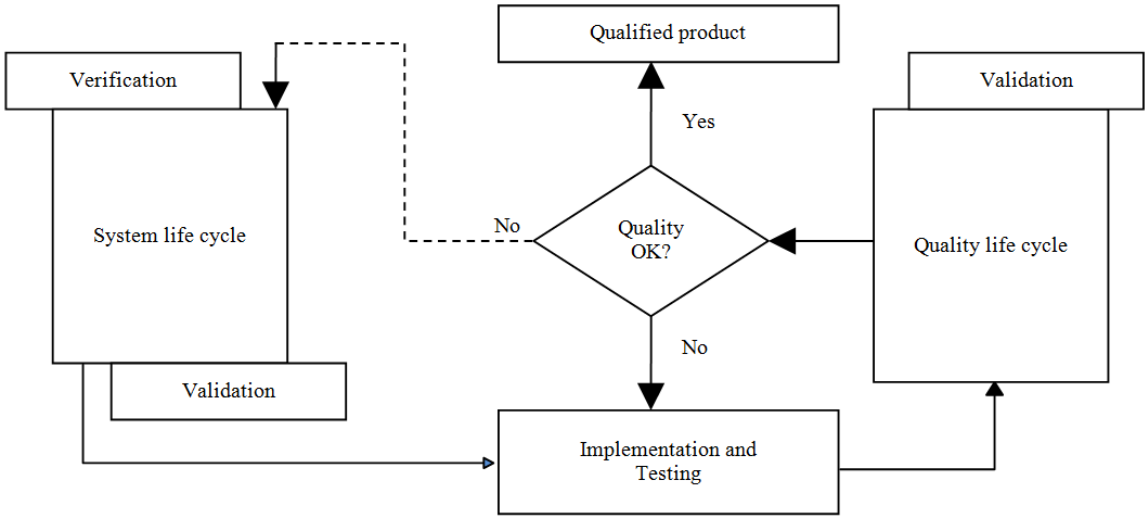

Fig. 2: Architectural U-model cycle for implementing quality life cycle

Instead of moving down in a liner way, the process steps are bent upwards after the coding phase, to form the typical u-shape. U-model demonstrates the relationships between each phase of the development of life cycle and its associated phase of testing. The horizontal and vertical axes represents time or project completeness (left-to-right) and level of abstraction (coarsest-grain abstraction uppermost), respectively.

This model is always better to introduce testing in early phases of Software Development Life Cycle. (SDLC), Lifecycle starts with the identification of a requirement for software and ends with the verification of the developed software against that requirement. Usually, SDLC models are used sequential, with the development progressing through a number of well defined phases. The U Lifecycle phases are usually represented by a U diagram. In the Requirements Analysis phase, it will be gathered the needs of the user, to produce a complete and unambiguous specification of the software. In the Design phase, the phase of the design of computer architecture and software architecture can also be referred to as high level design. The baseline in selecting the architecture is that it should realize all which typically consist of the list of module. This phase identifying the components within the software and the relationship between the components and it will implement detailed information of each component. The Code and Unit Test Phase, will implement various component of the software is coded and tested to verify that implement the detailed design. The Software Integration phase, it will implements larger group of tested software components are integrated and tested until the software works correctly and integrated with the system to check overall product. The Acceptance Testing phase is checking that the system delivered what was requested, (Fig. 3).

Software specification will be products of the first phases of this U Lifecycle model and the remaining phases are totally involve on testing at various levels.

SOFTWARE QUALITY

Quality software is sensible bug-free; consigned on time and within budget, encounters requirements and/or anticipations and is maintainable. However, quality is conspicuously a personal term. It will depend on who the ‘customer’ is and their overall leverage in the design of things. A wide-angle view of the ‘customers’ of a software development project might encompass end-users, customer acceptance testers, customer contract agents, customer management, the development organization's management/accountants/ testers/salespeople, future software upkeep engineers, stack holders, publication columnists, etc., each kind of ‘customer’ will have their own slant on ‘quality’ -the accounting department might define value in terms of profits while an end-user might characterize quality as user-friendly and bug-free, (Fig. 3).

QA team leader: Coordinates the testing activity, communicates testing status to manage and organize to the test team.

Software tester develop: Test script, test cases and data, script execution, metrics analysis and outcomes evaluation for system, integration and regression testing. Organizations alter substantial in how they accredit blame for QA and testing. Sometimes they are the blended responsibility of one assembly or one-by-one. Further more common are project teams that include a blend of testers and developers who work nearly together, with general QA methods supervised by project managers. It will count on what best aligns an organization's dimensions and business structure.

QA ensures that all parties worried with the project adhere to the process and procedures, measures and templates and test readiness reconsiders, (Fig. 4).

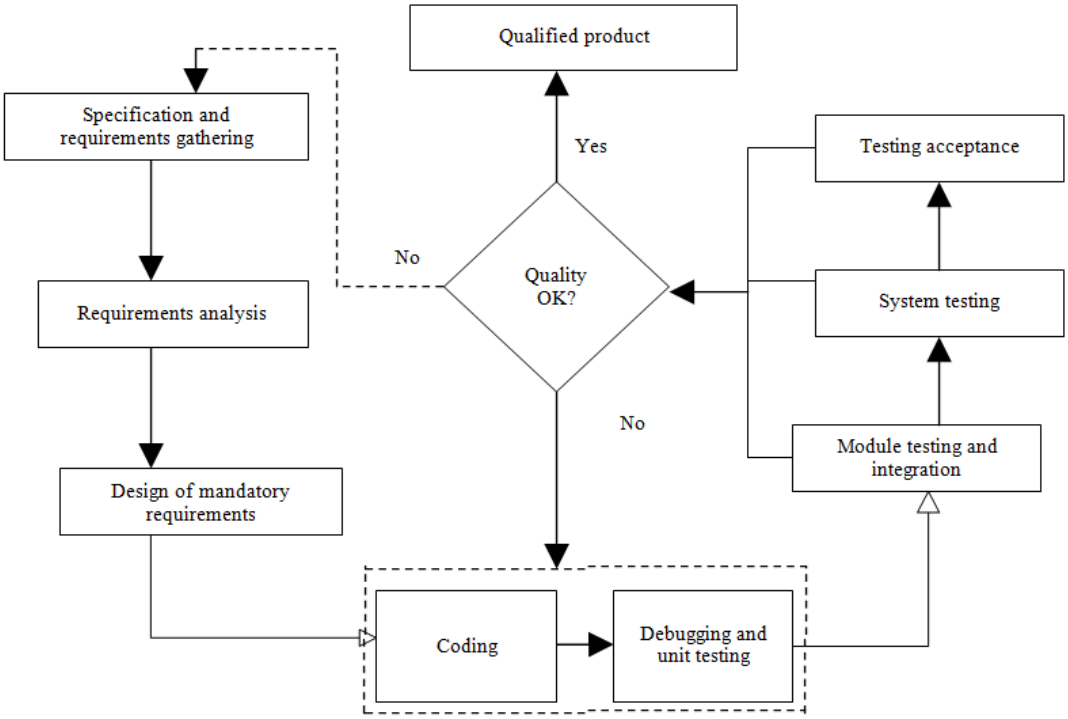

Fig. 3: Detailed U-model of testing and quality implementation

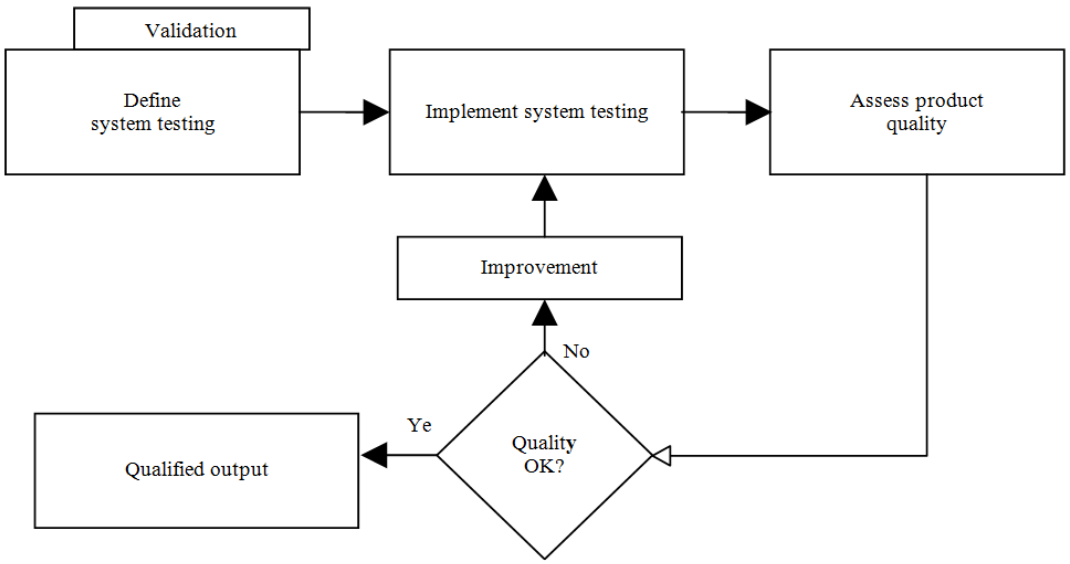

Fig. 4: Phase-based quality

Our QA service counts on the customers and projects. Allotment will count on team directs or managers, feedback to developers and double-checking ample communications amidst customers, managers, developers and testers.

RESULTS AND DISCUSSION

Quality implementation: Overall goal of Quality implementation is to deliver software with minimizes defects and meets expected levels of function, reliability and performance. Quality implementation makes sure that the project will be completed based on the agreed specifications, standards and functionality without defects and possible problems. The main benefits required are to do the effective testing before production deployments are to find defect before an application and before impact business operations. This reduces business disruptions reduces the cost of fixing of the defects from software failure or errors, (Fig. 4).

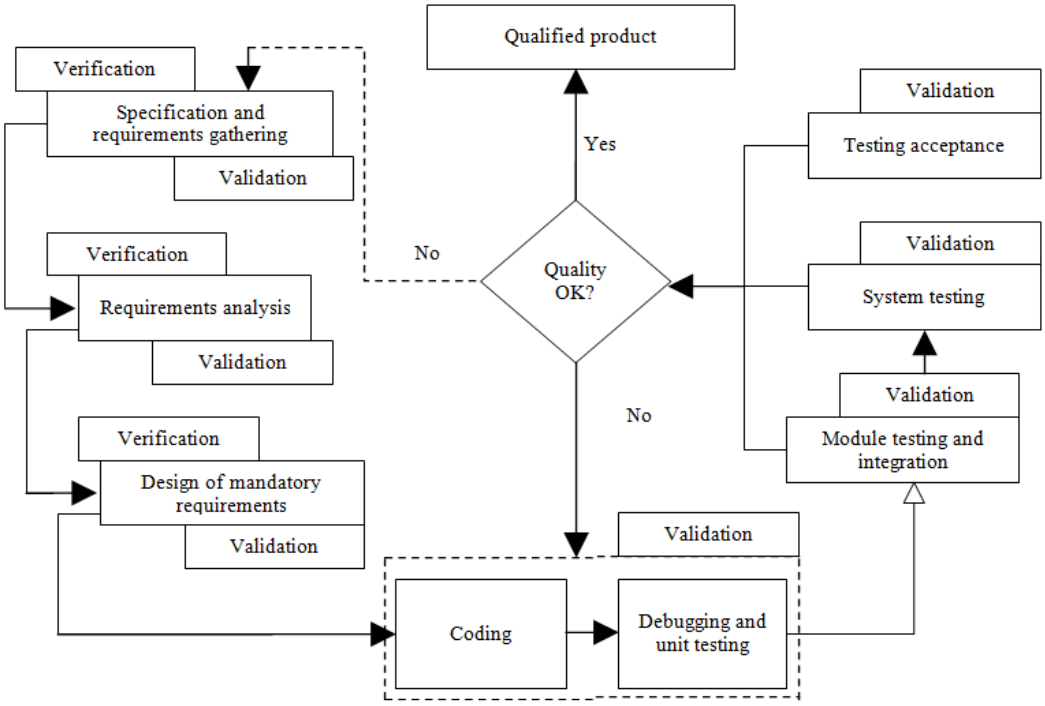

Fig. 5: Complete U-model with V and V and quality implementation

Count undiscovered defect in the software with estimation to decide when the software meets reliability criteria for production deployment.

Test result will help to identify strengths and deficiencies in development process and improvements that improve delivered software. The value of software checking is that is goes for after checking the underlying code.

It furthermore examines the functional behavior of the application. Behavior is a function of the code, but it doesn't habitually follow that if the behavior is "bad" then the code is awful. It's solely possible that the code is solid but the obligations were inaccurately or incompletely assembled and communicated, (Fig. 5).

Measurement endows the organization to advance the software method; assist in designing, tracking and commanding the programs project and assess the quality of the programs therefore produced. It is the assess of such exact attributes of the process, task and merchandise that are used to compute the programs metrics. Metrics are analyzed and they provide a dashboard to the administration on the general wellbeing of the process, task and product. Generally account for if the quality requirements have been achieved or are expected to be accomplished during the software development process.

As a quality assurance process, a metric is required to be revalidated every time it is used. These encompass the user satisfaction and software acceptability with their distinct dimensions which are capability or functionality, usability, performance, reliability and maintainability. In general, for most software quality assurance systems the common software metrics that are checked for enhancement are the source lines of code, cyclomatic complexity of the code, Function point analysis, bugs per line of code, code treatment, number of classes and interfaces, cohesion and coupling between the modules etc. widespread programs metrics include: Bugs per line of code, Code coverage, Cohesion, Coupling, Cyclomatic complexity, Function point analysis, Number of classes and interfaces, Number of lines of customer requirements, Order of growth and Source lines of code, (Fig. 5).

Software quality metrics aim on the process, project and product. By investigating the metrics the association can take corrective action to rectify those localities in the process, project or product which are the cause of the software defects. The de-facto definition of software quality comprises of the two major attributes based on intrinsic product quality and the client acceptability, (Fig. 6).

Maintainability: Is the ease with which a program can be correct if a mistake occurs. Since there is no direct way of assessing this indirect way has been used to assess this. It assesses when a mistake is discovered. How much time it takes to investigate the change, design the modification, implement it and check it.

Integrity: Measures the system's ability to with stand attacks to its security.

Usability: Is how working is your software application? This significant attribute of your

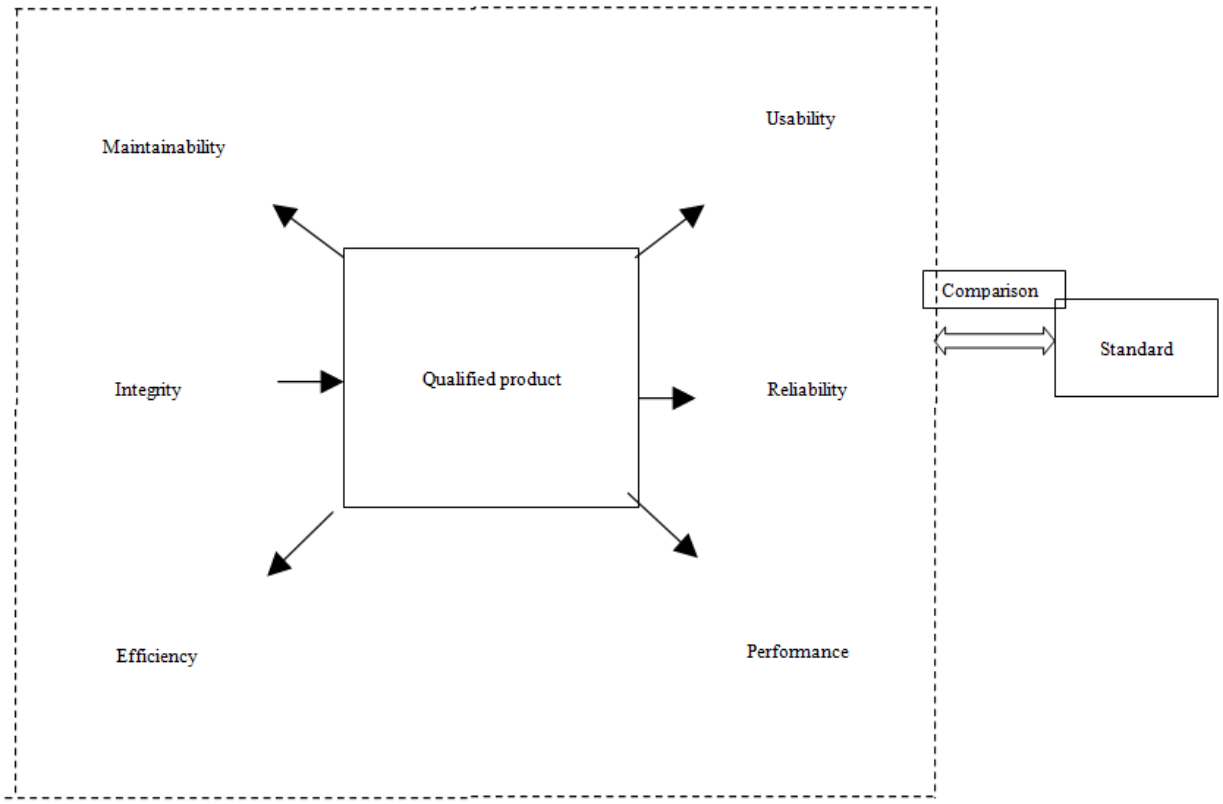

Fig. 6: Quality attributes comparison with system standards

submission is assessed in periods of the Time needed to become quite effective in the system, the snare boost in productivity by use of the system and subjective assessment (usually in the pattern of questionnaire on the new system).

Performance (efficiency): Has been a driving factor in systems architecture and often compromises the achievement of other quality attributes.

Reliability: Is the ability of a system to remain operational over time. Reliability is measured as the probability that a system will not fail to perform its intended functions over a specified time interval.

CONCLUSION

This study has applying quality assurance requirements after each of the different testing stages: unit testing, module testing sub system testing and system testing may lead to a qualified tested system. This will absolutely reduce the time, the cost and the effort of maintenance phase which considered as a major challenge of Software engineering discipline. Our study introduces a new dynamic model for software testing quality which embraces the quality assurance properties among different testing sub-phases.

REFERENCES

Aggarwal, K.K. and S. Yogesh, 2005. Software Engineering. 2nd Edn., New Age International.

Aranda, J., S. Easterbrook and G. Wilson, 2007. Requirements in the wild: How small companies do it. Proceeding of the 15th IEEE International Requirements Engineering Conference (RE 2007).

Chandrasehakhar, K., 2005. Software Engineering and Quality Assurance. BPB.

Dan, P. and M. Russ, 2008. Head First Software Development. 1st Edn., O'Reilly, Sebastopol, CA.

David, J. and S. Hossein, 2005. Test-driven development: Concepts, taxonomy and future directions. IEEE Software, 38(9): 43-50.

Gottesdiener, E., 2005. The Software Requirements Memory Jogger: A Pocket Guide to Help Software and Business Teams Develop and Manage Requirements. GOAL/QPC, Salem, NH.

James, A.W., 2009. Exploratory Software Testing: Tips, Tricks, Tours and Techniques to Guide Test Design. Addison Wesley, ISBN-10: 0321636414; ISBN-13: 978-0321636416.

Karl, E.W., 2006. More About Software Requirements: Thorny Issues and Practical Advice. Microsoft Press, Redmond, WA.

Lewis, W., 2008. Software Testing and Continuous Quality Improvement. Auerbach, Boca Raton, FL.

Pekka, A., B. Nathan, M. Tiziana and M. Richard, 2007. Software process improvement. Proceeding of the Software Process Model 14th European Conference (EuroSPI 2007). Potsdam, Germany, September 26-28, 2007.

Pine, F.J., F. García and M. Piattini, 2008. Software process improvement in small and medium software enterprisesa systematic review. Softw. Qual. Control, 16(2): 237-261.

Pressman, R.S., 2005. Software Engineering: A Practitioner's Approach. McGraw-Hill, New York.

Pressman, R.S. and D. Ince, 2007. Software Engineering-a Practitioner's Approach. McGraw-Hill.

Roger, S.P., 2005. Software Engineering: A Practioner's Approach. 6th Edn., (International Edn.), McGraw-Hill, NY.

Schulmeyer, G.G., 2007. Handbook of Software Quality Assurance. 4th Edn., Artech House, Boston.

Software Engineering Institute, 2008. The Ideal Model. Retrieved from: www: http:/ /www. sei. cmu. edu/ ideal/. (Accessed on: October 10, 2008)

Thayer, R.H. and M.J. Christiansen, 2005. Software Engineering. Volume 1: The Development Process. 3rd Edn., Wiley and Sons, NY.

Timothy, C.L. and L. Robert, 2005. Object-Oriented Software Engineering: Practical Software Development using UML and Java. 2nd Edn., McGraw-Hill, NY.

Weinberg, G.M., 2008. Perfect Software: And Other Illusions about Testing. Dorset House Publishing, New York.