Enhancing Software Quality by an SPL Testing based Software Testing

Hosam F. El-Sofany[1], Islam A. Taj-Eddin[2], Hassan El-Hoimal[1],

Tourki Al-Tourki[1], and Amjad Al-Sadoon[1]

[1]Arab East Colleges for Graduate Studies, Department of Computer Science, Riyadh, Kingdom of Saudi Arabia
[2] Faculty of Informatics and Computer Science British University in Egypt-BUE, Cairo, Egypt

ABSTRACT

Software testing is a process of providing information that a software application or program meets the business and technical requirements that guided its design and development; and works as expected. Regardless of the environment (i.e. Structured or Object Oriented), software testing can also identifies important defects, flaws, or errors in the application code that must be fixed. Quality control involves the maintenance of quality in a product, particularly by comparing the performance of a product with a set of specifications. This paper introduces some basic knowledge regarding to software testing. The paper provides some new model that could improve quality of software testing as a mean to increase the overall quality of the product.

Keywords

Software testing, Software quality, Testing strategy, Software quality assurance, software quality control.

1. INTRODUCTION

The purpose of software testing is not only to detect when problems occur, but also to help in diagnosing the nature of the problem. Quality control involves the maintenance of quality in a product, particularly by comparing the performance of a product with a set of specifications. When the product is software, an important part of quality control is software testing, where the software is run to determine whether it produces the desired output [12, 13].

Most people are confused with the concepts and difference between Quality Assurance, Quality Control and Testing. Although they are interrelated and at some level they can be considered as the same activities, but there is indeed a difference between them. Mentioned below are the definitions and differences between them [16-18]:

Quality Assurance	Quality Control	Testing
Activities which ensure the implementation of processes, procedures and standards in context to verification of developed software and intended requirements.	Activities which ensure the verification of developed software with respect to documented (or not in some cases) requirements.	Activities which ensure the identification of bugs/error/defects in the Software.
Focuses on processes and procedures rather than conducting actual testing on the system.	Focuses on actual testing by executing Software with intend to identify bug/defect through implementation of procedures and process.	Focuses on actual testing.
Process oriented activities.	Product oriented activities.	Product oriented activities.
Preventive activities.	Corrective process.	Corrective process.
A subset of Software Test Life Cycle (STLC).	Can be considered as the subset of Quality Assurance.	Subset of Quality Control

The main objective of this paper is to introduce an enhanced software quality model and software testing strategy to test the different factors that effect on the quality of software as well as increasing the productivity of the software by taking in consideration the software complexity that can faces the designer and implementer of the software systems.

The paper is organized as follows: in section two, we present the software testing concepts and definitions. In section three, we present a brief survey and a literature review, about the software testing and software quality. In section four, we introduce some methods for improving the quality of software testing. In section five, we introduce an enhanced model to provide software quality assurance. The paper finally concluded in section six.

2. TESTING BASIC CONCEPTS

2.1 Software Testing

Software testing is the process of executing software in a controlled manner, in order to answer the question: Does the software behave as specified?. Software testing is often used in association with the terms *verification* and *validation*.

Verification: It answers the question: Are we doing the job right?. It is the process to make sure the product satisfies the conditions imposed at the start of the development phase.

Validation: It answers the question: Are we doing the right job?. It is the process to make sure the product satisfies the specified requirements at the end of the development phase.

Software testing should not be confused with debugging. Debugging is the process of analyzing and locating bugs when software does not behave as expected. The term **bug** is often used to refer to a problem or fault in a computer. There are software bugs and hardware bugs. The term originated in the United States, at the time when pioneering computers were built out of valves, when a series of previously inexplicable faults were eventually traced to moths flying about inside the computer [16-18].

2.2 Software Testing Methods

There are different methods which can be used for Software testing. This section briefly describes those methods [16-18]:

I. Black Box Testing

The technique of testing without having any knowledge of the interior workings of the application is Black Box testing. The tester is oblivious to the system architecture and does not have access to the source code. Typically, when performing a black box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.

Advantages	Disadvantages
▪ Well suited and efficient for large code segments. ▪ Code Access not required. ▪ Clearly separates user's perspective from the developer's perspective through visibly defined roles. ▪ Large numbers of moderately skilled testers can test the application with no knowledge of implementation, programming language or operating systems.	▪ Limited Coverage since only a selected number of test scenarios are actually performed. ▪ Inefficient testing, due to the fact that the tester only has limited knowledge about an application. ▪ Blind Coverage, since the tester cannot target specific code segments or error prone areas. ▪ The test cases are difficult to design.

II. White Box Testing

White box testing is the detailed investigation of internal logic and structure of the code. White box testing is also called glass testing or open box testing. In order to perform white box testing on an application, the tester needs to possess knowledge of the internal working of the code.

The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately.

Advantages	Disadvantages
▪ As the tester has knowledge of the source code, it becomes very easy to find out which type of data can help in testing the application effectively. ▪ It helps in optimizing the code. ▪ Extra lines of code can be removed which can bring in hidden defects. ▪ Due to the tester's knowledge about the code, maximum coverage is attained during test scenario writing.	▪ Due to the fact that a skilled tester is needed to perform white box testing, the costs are increased. ▪ Sometimes it is impossible to look into every nook and corner to find out hidden errors that may create problems as many paths will go untested. ▪ It is difficult to maintain white box testing as the use of specialized tools like code analyzers and debugging tools are required.

III. Grey Box Testing

Grey Box testing is a technique to test the application with limited knowledge of the internal workings of an application. In software testing, the term *the more you know the better* carries a lot of weight when testing an application.

Mastering the domain of a system always gives the tester an edge over someone with limited domain knowledge. Unlike black box testing, where the tester only tests the application's user interface, in grey box testing, the tester has access to design documents and the database. Having this knowledge, the tester is able to better prepare test data and test scenarios when making the test plan.

Advantages	Disadvantages
▪ Offers combined benefits of black box and white box testing wherever possible. ▪ Grey box testers don't rely on the source code; instead they rely on interface definition and functional specifications. ▪ Based on the limited information available, a grey box tester can design excellent test scenarios especially around communication protocols and data type handling. ▪ The test is done from the point of view of the user and not the designer.	▪ Since the access to source code is not available, the ability to go over the code and test coverage is limited. ▪ The tests can be redundant if the software designer has already run a test case. ▪ Testing every possible input stream is unrealistic because it would take an unreasonable amount of time; therefore, many program paths will go untested.

IV. Black Box vs Grey Box vs White Box

Black Box Testing	Grey Box Testing	White Box Testing
The Internal Workings of an application are not required to be known	Somewhat knowledge of the internal workings are known	Tester has full knowledge of the Internal workings of the application
Also known as closed box testing, data driven testing and functional testing	Another term for grey box testing is translucent testing as the tester has limited knowledge of the insides of the application	Also known as clear box testing, structural testing or code based testing
Performed by end users and also by testers and developers	Performed by end users and also by testers and developers	Normally done by testers and developers
Testing is based on external expectations - Internal behavior of the application is unknown	Testing is done on the basis of high level database diagrams and data flow diagrams	Internal workings are fully known and the tester can design test data accordingly
This is the least time consuming and exhaustive	Partly time consuming and exhaustive	The most exhaustive and time consuming type of testing
Not suited to algorithm testing	Not suited to algorithm testing	Suited for algorithm testing
This can only be done by trial and error method	Data domains and Internal boundaries can be tested, if known	Data domains and Internal boundaries can be better tested

2.3 Levels of Software Testing

There are different levels during the process of Testing. In this section a brief description is provided about these levels.

Levels of testing include the different methodologies that can be used while conducting Software Testing. Following are the main levels of Software Testing: Functional Testing, and Non-Functional Testing.

I. Functional Testing

This is a type of black box testing that is based on the specifications of the software that is to be tested. The application is tested by providing input and then the results are examined that need to conform to the functionality it was intended for. Functional Testing of the software is conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements.

There are five steps that are involved when testing an application for functionality.

	Description
1	The determination of the functionality that the intended application is meant to perform.
2	The creation of test data based on the specifications of the application.
3	The output based on the test data and the specifications of the application.
4	The writing of Test Scenarios and the execution of test cases.
5	The comparison of actual and expected results based on the executed test cases.

An effective testing practice will see the above steps applied to the testing policies of every organization and hence it will make sure that the organization maintains the strictest of standards when it comes to software quality.

A. Unit Testing

This type of testing is performed by the developers before the setup is handed over to the testing team to formally execute the test cases. Unit testing is performed by the respective developers on the individual units of source code assigned areas. The developers use test data that is separate from the test data of the quality assurance team.

The goal of unit testing is to isolate each part of the program and show that individual parts are correct in terms of requirements and functionality. In an Object-Oriented environment, this is usually done at the class level. The minimal unit tests include the constructors and destructors [15].

Testing cannot catch each and every bug in an application. It is impossible to evaluate every execution path in every software application. The same is the case with unit testing.

There is a limit to the number of scenarios and test data that the developer can use to verify the source code. So after he has exhausted all options there is no choice but to stop unit testing and merge the code segment with other units.

B. Integration Testing

The testing of combined parts of an application to determine if they function correctly together is Integration testing. There are two methods of doing Integration Testing Bottom-up Integration testing and Top Down Integration testing.

- **Bottom-up integration:** This testing begins with unit testing, followed by tests of progressively higher-level combinations of units called modules or builds.
- **Top-Down integration:** This testing, the highest-level modules are tested first and progressively lower-level modules are tested after that.

In a comprehensive software development environment, bottom-up testing is usually done first, followed by top-down testing. The process concludes with multiple tests of the complete application, preferably in scenarios designed to mimic those it will encounter in customers' computers, systems and network [15, 16].

C. System Testing

Once all the components are integrated, the application as a whole is tested rigorously to see that it meets Quality Standards. This type of testing is performed by a specialized testing team.

System testing is so important because of the following reasons [15-18]:

- System Testing is the first step in the Software Development Life Cycle, where the application is tested as a whole.
- The application is tested thoroughly to verify that it meets the functional and technical specifications.
- The application is tested in an environment which is very close to the production environment where the application will be deployed.
- System Testing enables us to test, verify and validate both the business requirements as well as the Applications Architecture.

D. Regression Testing

Whenever a change in a software application is made it is quite possible that other areas within the application have been affected by this change. To verify that a fixed bug hasn't resulted in another functionality or business rule violation is Regression testing. The intent of Regression testing is to ensure that a change, such as a bug fix did not result in another fault being uncovered in the application.

Regression testing is so important because of the following reasons:

- Minimize the gaps in testing when an application with changes made has to be tested.
- Testing the new changes to verify that the change made did not affect any other area of the application.
- Mitigates Risks when regression testing is performed on the application.
- Test coverage is increased without compromising timelines.
- Increase speed to market the product

E. Acceptance Testing

This is arguably the most importance type of testing as it is conducted by the Quality Assurance Team who will gauge whether the application meets the intended specifications and satisfies the client's requirements. The QA team will have a set of pre written scenarios and Test Cases that will be used to test the application.

More ideas will be shared about the application and more tests can be performed on it to gauge its accuracy and the reasons why the project was initiated. Acceptance tests are not only intended to point out simple spelling mistakes, cosmetic errors or Interface gaps, but also to point out any bugs in the application that will result in system crashers or major errors in the application.

By performing acceptance tests on an application the testing team will deduce how the application will perform in production. There are also legal and contractual requirements for acceptance of the system [16-18].

F. Alpha Testing

This test is the first stage of testing and will be performed amongst the teams (developer and QA teams). Unit testing, integration testing and system testing when combined are known as alpha testing. During this phase, the following will be tested in the application:

- Spelling Mistakes
- Broken Links
- Cloudy Directions
- The Application will be tested on machines with the lowest specification to test loading times and any latency problems.

G. Beta Testing

This test is performed after Alpha testing has been successfully performed. In beta testing a sample of the intended audience tests the application. Beta testing is also known as pre-release testing. Beta test versions of software are ideally distributed to a wide audience on the Web, partly to give the program a "real-world" test and partly to provide a preview of the next release. In this phase the audience will be testing the following:

- Users will install, run the application and send their feedback to the project team.
- Typographical errors, confusing application flow, and even crashes.
- Getting the feedback, the project team can fix the problems before releasing the software to the actual users.
- The more issues you fix that solve real user problems, the higher the quality of your application will be.
- Having a higher-quality application when you release to the general public will increase customer satisfaction.

II. Non-Functional Testing

Non-functional testing of Software involves testing the Software from the requirements which are non functional in nature such as performance, security, user interface etc. Some of the important and commonly used non-functional testing types are [16-18]:

A. Performance Testing

It is mostly used to identify any bottlenecks or performance issues rather than finding the bugs in software. There are different causes which contribute in lowering the performance of software: Network delay, client side processing, database transaction processing, load balancing between servers, and data rendering.

Performance testing is considered as one of the important and mandatory testing type in terms of following aspects: Speed, capacity, stability, and Scalability.

It can be either qualitative or quantitative testing activity and can be divided into different sub types such as*: load testing* and *Stress testing*.

- ***Load*** **Testing:** A process of testing the behavior of the Software by applying maximum load in terms of Software accessing and manipulating large input data. It

can be done at both normal and peak load conditions. This type of testing identifies the maximum capacity of Software and its behavior at peak time. Most of the time, Load testing is performed with the help of automated tools such as Load Runner, AppLoader, IBM Rational Performance Tester, Apache JMeter, Silk Performer, Visual Studio Load Test etc. Virtual users (VUsers) are defined in the automated testing tool and the script is executed to verify the Load testing for the Software. The quantity of users can be increased or decreased concurrently or incrementally based upon the requirements.

- **Stress Testing*:*** This testing type includes the testing of Software behavior under abnormal conditions. Taking away the resources, applying load beyond the actual load limit is Stress testing. The main intent is to test the Software by applying the load to the system and taking over the resources used by the Software to identify the breaking point. This testing can be performed by testing different scenarios such as:
    - Shutdown or restart of Network ports randomly.
    - Turning the database on or off.
    - Running different processes that consume resources such as CPU, Memory, server etc.

B. Usability Testing

This section includes different concepts and definitions of usability testing from Software point of view. It is a black box technique and is used to identify any error(s) and improvements in the Software by observing the users through their usage and operation [16].

- Usability can be defined in terms of five factors *i.e*. Efficiency of use, Learn-ability, Memory-ability, Errors/safety, satisfaction. According to this definition, the usability of the product will be good and the system is usable if it possesses the above factors.
- We can consider that usability is the quality requirement which can be measured as the outcome of interactions with a computer system. This requirement can be fulfilled and the end user will be satisfied if the intended goals are achieved effectively with the use of proper resources.
- Other authors stated that user friendly system should fulfill the following five goals (i.e. Easy to Learn, Easy to Remember, Efficient to Use, Satisfactory to Use and Easy to understand).

In addition to different definitions of usability, there are some standards and quality models and methods which define the usability in the form of attributes and sub attributes such as ISO-9126, ISO-9241-11, ISO-13407 and IEEE std.610.12 etc.

C. Graphical User Interface Testing

User interface (UI) testing involves the testing of Graphical User Interface of the Software. This testing ensures that the GUI should be according to requirements in terms of color, alignment, size and other properties.

On the other hand Usability testing ensures that a good and user friendly GUI is designed and is easy to use for the end user. UI testing can be considered as a sub part of Usability testing.

D. Security Testing

Security testing involves the testing of Software in order to identify any flaws ad gaps from security and vulnerability point of view. Following are the main aspects which Security testing should ensure: confidentiality, integrity, authentication, availability, authorization, non-repudiation, software data is secure, software is according to all security regulations, input checking and validation, SQL insertion attacks, injection flaws, session management issues, cross-site scripting attacks, buffer overflows vulnerabilities, and directory traversal attacks.

E. Portability Testing

Portability testing includes the testing of Software with intend that it should be re-useable and can be moved from another Software as well. Following are the strategies that can be used for Portability testing [16-18].

- Transferred installed Software from one computer to another.
- Building executable (.exe) to run the Software on different platforms.

Portability testing can be considered as one of the sub parts of System testing, as this testing type includes the overall testing of Software with respect to its usage over different environments. Computer Hardware, Operating Systems and Browsers are the major focus of Portability testing. Following are some pre-conditions for Portability testing:

- Software should be designed and coded, keeping in mind Portability Requirements.
- Unit testing has been performed on the associated components.
- Integration testing has been performed.
- Test environment has been established.

2.4 Software Specifications Documents and Testing

A hierarchy of software specifications will typically contain three or more levels of software specification documents, see Figure 1 [16].

- The **Requirements Specification**, which specifies what the software is required to do and may also specify constraints on how this may be achieved.
- The **Architectural Design Specification**, which describes the architecture of a design which implements the requirements. Components within the software and the relationship between them will be described in this document.
- **Detailed Design Specifications**, which describe how each component in the software, down to individual units, is to be implemented.

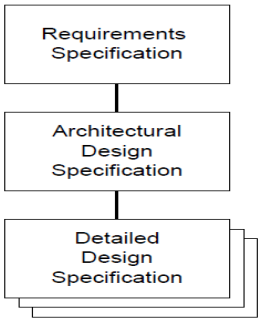

Fig 1: A hierarchy of software specifications

With this hierarchy of specifications, it is possible to test software at various stages of the development, for conformance with each specification. The levels of testing which correspond to the hierarchy of software specifications listed above are: Unit testing, integration testing, System testing, and Acceptance testing. All four had been explained previously.

Testing does not end following the conclusion of acceptance testing. Software has to be maintained to fix problems which show up during use and to accommodate new requirements. Software tests have to be repeated, modified and extended. regression testing is of use for this case. It had been explained previously. The effort to revise and repeat tests consequently forms a major part of the overall cost of developing and maintaining software.

Once each level of software specification has been written, the next step is to design the tests. The design of tests is subject to the same basic engineering principles as the design of software. Good design consists of a number of stages which progressively elaborate the design of tests from an initial high level strategy to detailed test procedures. These stages are: test strategy, test planning, test case design, and test procedure design.

The design of tests has to be driven by the specification of the software. At the highest level this means that tests will be designed to verify that the software faithfully implements the requirements of the Requirements Specification. At lower levels tests will be designed to verify that items of software implement all design decisions made in the Architectural Design Specification and Detailed Design Specifications. As with any design process, each stage of the test design process should be subject to informal and formal review.

The ease with which tests can be designed is highly dependent on the design of the software. It is important to consider testability as a key (but usually undocumented) requirement for any software development.

The following steps can help in test design and documented [16].

I. **Test Strategy:** A test strategy is a statement of the overall approach to testing, identifying what levels of testing are to be applied and the methods, techniques and tools to be used. A test strategy should ideally be organization wide, being applicable to all of an organizations software development. Developing a test strategy which efficiently meets the needs of an organization is critical to the success of software development within the organization. The application of a test strategy to a software development project should be detailed in the projects *software quality plan.*

II. **Test Plans:** The next stage of test design, which is the first stage within a software development project, is the development of a *test plan.* A test plan states what the items to be tested are, at what level they will be tested, what sequence they are to be tested in, how the test strategy will be applied to the testing of each item, and describes the test environment. A test plan may be project wide, or may in fact be a hierarchy of plans relating to the various levels of specification and testing:

- An *Acceptance Test Plan*, describing the plan for acceptance testing of the software. This would usually be published as a separate document, but might be published with the system test plan as a single document.
- A *System Test Plan*, describing the plan for system integration and testing. This would also usually be published as a separate document, but might be published with the acceptance test plan.
- A *Software Integration Test Plan*, describing the plan for integration of tested software components. This may form part of the Architectural Design Specification.
- *Unit Test Plan***(s)**, describing the plans for testing of individual units of software. These may form part of the Detailed Design Specifications.

The objective of each test plan is to provide a plan for verification, by testing the software, that the software produced fulfils the requirements or design statements of the appropriate software specification. In the case of acceptance testing and system testing, this means the Requirements Specification.

III. **Test Case Design:** Once the test plan for a level of testing has been written, the next stage of test design is to specify a set of *test cases* or *test paths* for each item to be tested at that level. A number of test cases will be identified for each item to be tested at each level of testing. Each test case will specify how the implementation of a particular requirement or design decision is to be tested and the criteria for success of the test. The test cases may be documented with the test plan, as a section of a software specification, or in a separate document called a *test specification* or *test description.*

- An *Acceptance Test Specification*, specifying the test cases for acceptance testing of the software. This would usually be published as a separate document, but might be published with the acceptance test plan.
- A *System Test Specification*, specifying the test cases for system integration and testing. This would also usually be published as a separate document, but might be published with the system test plan.
- *Software Integration Test Specifications*, specifying the test cases for each stage of integration of tested software components. These may form sections of the *Architectural Design Specification.*
- *Unit Test Specifications*, specifying the test cases for testing of individual units of software. These

may form sections of the Detailed Design Specifications.

System testing and acceptance testing involve an enormous number of individual test cases. In order to keep track of which requirements are tested by which test cases, an index which cross references between requirements and test cases often constructed. This is usually referred to as a *Verification Cross Reference Index* (VCRI) and is attached to the test specification. Cross reference indexes may also be used with unit testing and software integration testing.

It is important to design test cases for both *positive testing* and *negative testing*. Positive testing checks that the software does what it should. Negative testing checks that the software doesn't do what it shouldn't.

The process of designing test cases, including executing them as *thought experiments*, will often identify bugs before the software has even been built. It is not uncommon to find more bugs when designing tests than when executing tests.

IV. **Test Procedures:** The final stage of test design is to implement a set of test cases as a test procedure, specifying the exact process to be followed to conduct each of the test cases. This is a fairly straight forward process, which can be likened to designing units of code from higher level functional descriptions.

For each item to be tested, at each level of testing, a test procedure will specify the process to be followed in conducting the appropriate test cases. A test procedure cannot leave out steps or make assumptions. The level of detail must be such that the test procedure is deterministic and repeatable.

V. Test Results: When tests are executed, the outputs of each test execution should be recorded in a *test results file*. These results are then assessed against criteria in the test specification to determine the overall outcome of a test. Each test execution should also be noted in *a test log*. The test log will contain records of when each test has been executed, the outcome of each test execution, and may also include key observations made during test execution. Often a test log is not maintained for lower levels of testing (unit test and software integration test). *Test reports* may be produced at various points during the testing process. Test reports will summaries the results of testing and document any analysis. An *acceptance test report* often forms a contractual document within which acceptance of software is agreed.

3. PREVIOUS WORK ON SPL TESTING AND QUALITY CONTROL

Every application or business domain faces a specific set of software quality issues, and software quality must be defined accordingly. It is important for each software development project to define its specific meaning of software quality during the planning phase. Such a definition contributes to the basis for setting objectives and practical measures of quality progress and determination of readiness for release to customers [1].

Reeves and Bednar [2] stated that "no universal, Parsimonious, or all-encompassing definition or model of quality exists". The American National Standards Institute (ANSI) and American Society for Quality (ANQ) define quality as: "The totality of features and characteristics of a product or service that impact its ability to satisfy given needs."

Bo Zhang, Changchun and Xiangheng Shen [5], defined Software testing is an approach to guarantee software quality, and with its effectiveness people can establish confidence in the correctness of software which has been previously tested.

Wesley K., Daniela de., Thelma E., Silvia R. [6] defined a Software Product Line (SPL) is a set of several software products that share a set of common features. These features meet specific needs of a particular domain and are developed from core assets previously defined. Reuse is the focus of the SPL approach. Differently from the classical reuse approach, where the software first is developed and packaged, and then the reuse may be considered, in the SPL approach, the development is oriented to obtain a common architecture that will be used in all products of the SPL.

Clements et al [7] say that SPL engineering comprises three essential phases: Domain Engineering, Application Engineering and SPL Management. In the Domain Engineering, a set of core assets is developed. It has the common artifacts to the whole SPL and it may include plains, requirements, design, tests and code. In this activity, a generic architecture is defined, then, the products of the SPL are developed by the instantiation of this architecture according to the features of the product. Each specific product is generated by a single combination of features. The SPL development leads to a productivity increase and a reduction of costs and delivery time. Then, the development of a product of the SPL occurs in two steps: the development of components to reuse, in the Domain Engineering, and the product development from reusable components, in the Application Engineering [8]. This makes the SPL testing activity more critical and complex than test in the classical development process.

According to Lamancha et al [9] there is no process defined for SPL testing. They also say that there are some guidelines available to perform this activity. SPL testing must involve the test of core assets, of specific products and their interactions [10]. To perform such tasks, the existing test techniques and methods have been used. However, a testing strategy to allow the application of those methods in the SPL context is necessary.

Tevanlinna et al [8] describe strategies to SPL testing. The traditional strategy named Product by Product (PbP) consists of testing each product of the SPL separately, without the reuse of test assets. The strategy named Reusable Asset Instantiation (RAI) creates test specifications based on the Domain Engineering, in order to predict all of the possible variabilities. In the Application Engineering phase, the test assets are refined and instantiate for the specific product under test. The main difference of both strategies is that the last one is in fact product line oriented. However, unnecessary test cases can be generated due to the great number of possible combinations considering the SPL variability's.

Strategies that are SPL oriented allow reuse of test assets and were proposed as an alternative to those ones that test product by product in a separated way. Considering this fact, the authors presented results of a case study to evaluate the test strategy, named RAI (Reusable Assets Instantiation). This strategy that is SPL oriented was compared with a traditional strategy PbP (Product by Product). The study used AGM, an SPL from the game context, and test data generation methods based on the use case model[6].

RAI considers that there is a common generic structure, and test assets can be reused for different products of SPL. This strategy allows the test starts at the begin of the Domain Engineering, because it uses the requirements as a source for generating the test specification. So it is not necessary to await the generation of products to start testing activities. when applying RAI, deriving all the domain test assets and application requirements needed to test the SPL. This methodology allows us to deal with the variability of SPL and instantiate test cases for a specific product[6].

The result of this case study has produced evidence that the RAI strategy is more suitable for SPL testing than the PbP strategy, considering reuse. Moreover, the case study constitutes an experience of SPL test which shows that the RAI strategy is effective [6].

4. NEW SOFTWARE QUALITY ASSURANCE MODEL

In [11] software complexity is added during the development stages that following the requirements phase, primarily during the designing and coding phase. In our case to provide software quality assurance, we performed the testing process after solution phase to ensure that design and code meet the requirements and to achieve a best quality before development phase, as shown in Figure 2, Also we have modified the model introduced in [11] by redesign the productivity phase that comes after quality phase to ensure produce software with high quality, fulfilled all the requirements, uncomplicated code, ease of maintenance and modification and less errors.

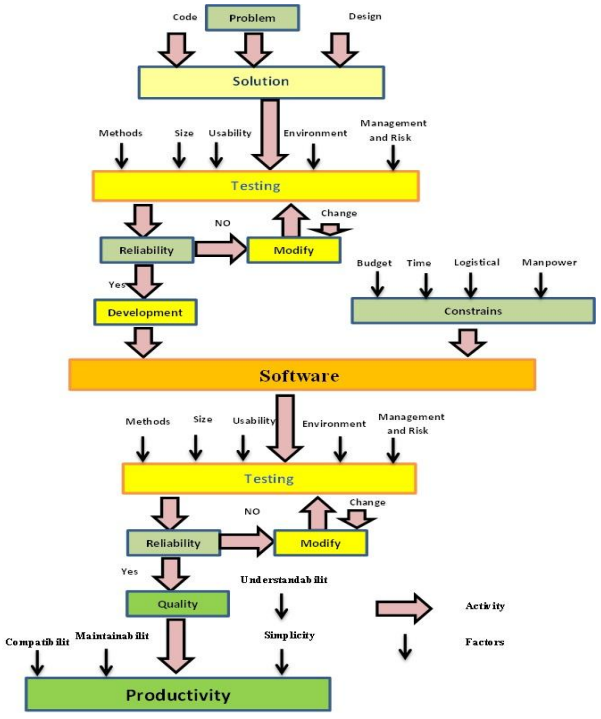

Fig 2: New SPL Testing Based Model for software quality assurance

5. CONCLUSIONS

In this paper we propose the difference between traditional testing and SPL testing strategy. The SPL testing strategy starts testing at the begin of development project, so we modified the old model based on that concept, because the old model do a testing after the finish from software development, we do the testing before and after the software development, which make the new model more easier to find the errors and modifying the code.

6. ACKNOWLEDGMENT

The authors acknowledge the full support received from Arab East Colleges for Graduate Studies, especially they very thankful to Professor Dr. Abd Allah Alfaisal, Dean of the Colleges, for his encouragement and support.

REFERENCES

[1] Lazic L., Kolasinac A., Avdic D., "The Software Quality Economics Model for Software Project Optimization", WSEAS Transaction on Computers, Issue 1, Vol. 8, January 2009.

[2] Reeves C., Bednar D., "Defining Quality: Alternative and Implications", Academy of Management Review 19 (3), pp. 419-445, 1994.

[3] ISO, International Organization for Standardization, "ISO 9000:2000, Quality Management Systems-Fundamentals and Vocabulary", 2000.

[4] ISO, International Organization for Standardization, "ISO 9004:2000, Quality Management Systems-Guidelines for performance improvements", 2000.

[5] Bo Zhang , Changchun and Xiangheng Shen, " The Effectiveness of Real-time Embedded Software Testing" , 978-1-61284-666-8/11$26.00 2011 IEEE.

[6] Wesley K., Daniela de., Thelma E., Silvia R. " Evaluating test reuse of a software product line oriented strategy", 978-1-4577-1490-01111$26.00 ©2011 IEEE

[7] P. C. Clements, L. G. Jones, J. D. Mcgregor and L. M. Northrop, "Getting There From Here: A Roadmap for Software Product Line Adoption". Communications of the ACM. Vol. 49, N° 12. Dec., 2006.

[8] A. Tevanlinna, J. Taina and R. Kauppinen. "Product Family Testing: a Survey". ACM SIGSOFT Sw Eng. Notes 29(2), 2004, pp. 1-6.

[9] B. P. Lamancha, M. Usaola and M. Piattini, "Software Product Line Testing, A systematic review". International Conference on Soft ware, vol. 49, 2009, pp. 78 - 81.

[10] J. D. McGregor. "Testing a soft ware product line". Technical Report CMU1SEI-2001-TR-022, Soft ware Engineering Institute, Carnegie Mellon University, 2001.

[11] Yas A. Alsultanny, and Ahmed M. Wohaishi, "*Requirements of Software Quality Assurance Model*", 2nd. International Conference on Environmental and Computer Science, DOI 10.1109/ICECS.2009.43, IEEE, 2009.

[12] Horch JW. Practical Guide to Software Quality Management. 2nd edition. Artech House Publishers. ISBN 9781580535274.

[13] Hower R. Software QA and Testing Frequently-Asked-Questions." http://www. Software qatest. Com /qatfaq 1.html. 2009.

[14] Basili, V.R. Qualitative software complexity models: A summary. In Tutorial on Models and Methods for Software Management and Engineering. IEEE Computer Society Press, Los Alamitos, Calif., 1980.

[15] Binder, Robert V. (1999). Testing Object-Oriented Systems: Objects, Patterns, and Tools. Addison-Wesley Professional. p. 45. ISBN 0-201-80938-9.

[16] http://www.tutorialspoint.com/software_testing/

[17] Software Testing Concepts and Practices, K. Mustafa and R.A. Khan. Alpha Science International Ltd. Oxford .U.K. 2007.

[18] Software Testing and Computing Quality Implementation, William Lewis, 2nd edition , 2004.