Software Testing Process Model from Requirement Analysis to Maintenance

Sukanta Basak
Department of ECE
North South University

Md. Shazzad Hosain
Department of ECE
North South University

ABSTRACT
It is infeasible to develop quality software without software testing. Software testing typically consumes 40-50% of development efforts, even more efforts for systems that require higher levels of reliability. It is a significant part of the software development process. If a software defect is found in latter stages of development process it costs more than if it is found in earlier stages of development. Thus the earlier we introduce testing, the less the defects found, which in turn reduces the development time and cost. Risk minimization is another approach for efficient software development. Traditionally risk management and test management are addressed separately, but if the two approaches were integrated it would further reduce time and cost of software development. There are a number of testing approaches and models for software development but no testing process model addresses defect prevention and risk minimization simultaneously. In this paper we propose a methodical or formal approach of software testing that introduces testing from the very beginning of software development life cycle as well as integrates risk management with test management. The proposed model has been evaluated in a number of software and it outperforms the existing models.

General Terms
Software engineering, software development life cycle and process model, software quality assurance.

Keywords
Software testing, validation, software risk management, software life cycle, unit testing, integration testing, system testing, regression testing.

1. INTRODUCTION
In order to build software not only bugs free but also to conform requirements it needs to be tested at several stages of its development life cycle. Due to thorough understanding of software engineering, tests become more and more important for software development. A study conducted by NIST in 2002 reports that software bugs cost the U.S. economy $59.5 billion annually. More than a third of this cost could be avoided if better software testing was performed [1]. Table 1 shows the cost of fixing defects depending on the stages it was found [2].

For example, if a problem in the requirements is found only at post-release phase, then it would cost 10–100 times more to fix than if it had already been identified by requirements review. Thus it is strongly recommended to identify requirement anomalies or to find faults at early stages of software development process.

Table 1. Average cost of fixing defects

Cost to fix a defect		Time detected				
		R	**A**	**C**	**System Test**	**Post-release**
Time introduced	**R**	1×	3×	5-10×	10×	10-100×
	A	-	1×	10×	15×	25-100×
	C	-	-	1×	10×	10-25×

R - Requirements, **A** – Architecture, **C** – Construction

Software process is a set of activities that are required to develop a software system. Typically the set consists of *specification*, *design*, *testing* and *evolution*. A good software process must have to address the following two issues:

a) Bug prevention so that fixing bugs at later phases does not increase the cost of development [3].

b) Risk minimization so that unwanted situations do not increase the delivery time [4].

A software process model describes the software process in some particular perspective. For example, in waterfall model the four steps of software process are done one after another. On the other hand in iterative models such as rational unified process (RUP), SCRUM [5] etc. the whole development phase is divided into different iterations or sprint in case of SCRUM, and in each iterations waterfall model is applied. In different software process models testing is introduced in different times of the development phase. For example iterative models introduce testing much earlier than waterfall model. The earlier we can introduce testing in software process the better the software. This encourages bug prevention [3]. Thus one of our goals is to introduce testing at the very beginning of software process.

Software risk management is concerned with identifying risks and drawing up plans to mitigate their impacts. Usually risk management and test management are viewed as two separate management issues though many risks prop up from improper test management. If we could integrate risk management and test management together then many risks would not appear, as a result software testing and development would be faster. The integration of risk and test management has been studied very recently in [4]. Thus another goal of our research is to integrate risk management and test management.

In the literature many different software testing approaches or models are found. Most of these models focus at one of more steps of software process. For example, specification-based testing focuses on functional testing, design-based testing focuses on data and process paths within the software structures, acceptance testing focuses on customer satisfaction and so on. All these testing approaches are described and used in isolation, but there is no methodical or formal way to use different testing approaches to build robust systems with least

cost. In this paper we propose a software testing process model that has the following advantages:

a) The proposed test process model addresses the whole life cycle of software development i.e. from requirement collection to maintenance.

b) It introduces testing at the early stage of software process such as in requirement analysis phase, thus it prevents bugs.

c) It incorporates risk management with test process, thus reduces the cost and effort for software testing.

d) The proposed testing model ensured early release of software.

The paper is organized as the following. Section 2 provides existing testing methodologies, section 3 describes the proposed software testing process model, section 4 gives evaluation and section 5 draws the conclusion.

2. TESTING METHODOLOGIES

To achieve the best result from software testing, it must be done in a methodical or formal approach. In formal approach software testing is done using some standard software testing steps. These steps in combination are known as software testing life-cycle. Software testing life-cycle starts and ends in different phases of software development life cycle depending on the process models.

In the field of software testing there are different types of software testing methodologies. Most commonly used software testing methodologies described here.

2.1 Post-Development Software Testing

In waterfall model, (requirement – design – coding – testing – integration – maintenance), testing work starts after the coding. Though informally programmers do unit tests during development, the major testing is done after development which is black box testing. Only quality assurance (QA) engineers are involved in this kind of testing. In this approach quality assurance engineers tests the functionality and behavior of software from the user's point of view rather than testing the internal structure of the software [6] [7]. In the old days of software testing this methodology was commonly used but now a days it is rarely used.

2.2 Software Testing in Iterative Models

Latest software development process use different iterative models such as rational unified process (RUP), spiral model, SCRUM and so on. In these development models the life cycle consists of a number of iterations, which is also known as sprint in SCRUM, where in each iteration waterfall model (analysis – design – coding – testing – integration – feedback) model is applied. Each iteration uses white-box testing (unit testing), then does integration testing and system testing. After that the added/updated functionalities are delivered to customers for acceptance testing [7]. Unlike waterfall model, where actual testing begins after coding or development of the entire software, it begins at the very early stage of development in iterative models. The system gets more opportunities to go under different kinds of tests. Thus defects are notified earlier, bugs are fixed and accepted by customers, project risk is lowered and overall cost is reduced.

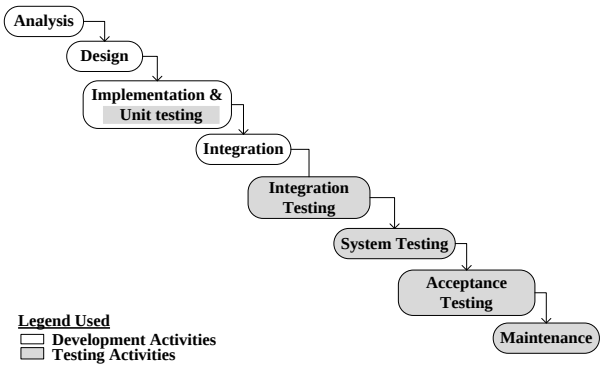

Fig 1: Testing activities in waterfall model.

2.3 Test Driven Software Development

In test driven development, test is written before writing the functionality as shown in Figure 2. For each new feature a test is written first. At this point the test must fail because the feature is not implemented yet. The next step is to write some code that will implement the feature and cause the test to pass. After implementing each new feature the developers run the entire test again. If all the tests pass then the implementation is all right. At this point the code is cleaned up as necessary. The process is repeated to implement other functionalities [8].

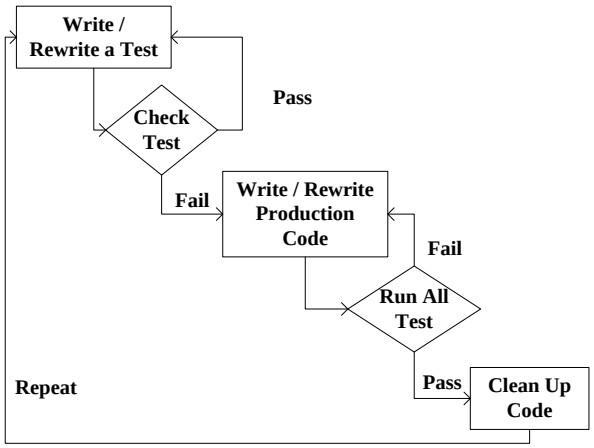

Fig 2: Test driven software development process

2.4 Limitations of Existing Methodologies

In the post-development software testing methodologies, testing starts after the coding. As a result the rate of faults in the implementation remains high. After the implementation fixing these faults becomes expensive.

The testing problem in waterfall model was significantly reduced in iterative models because software functionalities are delivered in steps and users were involved from the beginning. However, in iterative models verification & validation is not done before coding in the early phases of the development process. Faults found in the later phases of development process raise the cost and efforts of repairing process [1] [9]. Another big limitation of these methodologies is that they do not use any risk mitigation planning. Therefore all the areas need to be tested with equal importance. As a result they require more effort for testing software. The main drawback of test driven development is that it only covers the structural testing i.e. white-box testing. Therefore this testing methodology cannot cover the full development process. Other testing techniques still required to cover the full development life-cycle.

3. PROPOSED TESTING MODEL

Different software testing methodologies have been discussed with their limitations in the above section. Ideally testing should begin at the very early stage of software development. Though existing testing models facilitate testing in different phases of software life cycle, there is no model that took a holistic approach i.e. testing from requirement analysis to maintenance. In this paper we thus propose a holistic software testing model that has the following nine steps.

a) Test Team Building
b) Finalize Testing Objectives
c) Constructing the Test Plans
d) Test Cases Design and Construction
e) Unit Testing
f) Risk Assessment
g) Integration Testing
h) System Testing
i) Regression Testing

The steps are so arranged that these steps can fit into waterfall or iterative or any software process models. The steps are shown graphically in figure 3.

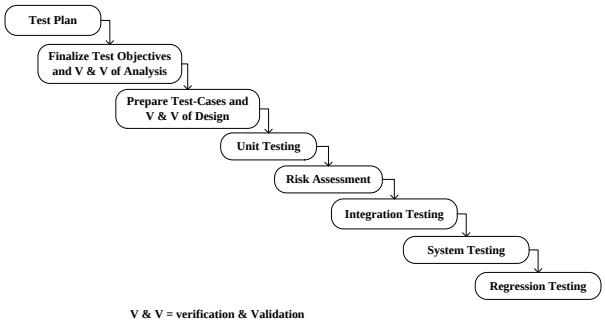

Fig 3: Proposed software testing process model.

In proposed testing methodology testing process will start from the very beginning i.e. analysis phase of the software development life cycle and will continue till maintenance level. Figure 4 shows testing activities of the proposed model that fit into waterfall model. Here, test plan construction starts in parallel to analysis phase and finishes at the early stage of that phase. The test plan describes the overall test strategies and test cases, task assignment, testing environments, testing schedules, error tracking & reporting etc. Then test objectives are finalized, after that verification and validation of analysis are started. Test objectives are statements of what the testers want to accomplish when implementing a specific testing activity. This process validates and verifies the software requirements which will ensure early detection of faults.

At the early state of the design phase test-case construction is started. These test cases are used in the later phases of software testing. After that design verification and validation starts and it is finished at the early stage of coding. Early detection of faults in design is ensured in this phase. Unit testing is done in parallel with coding. At the last stage of implementation, risk assessment is done. Risk assessment ensures appropriate testing in right areas and reduces the amount of testing effort in the post implementation phases. After that integration testing, system testing, regression testing and acceptance testing are done and the software goes into operation and maintenance.

The detail of the proposed software testing methodology is described in the following sections.

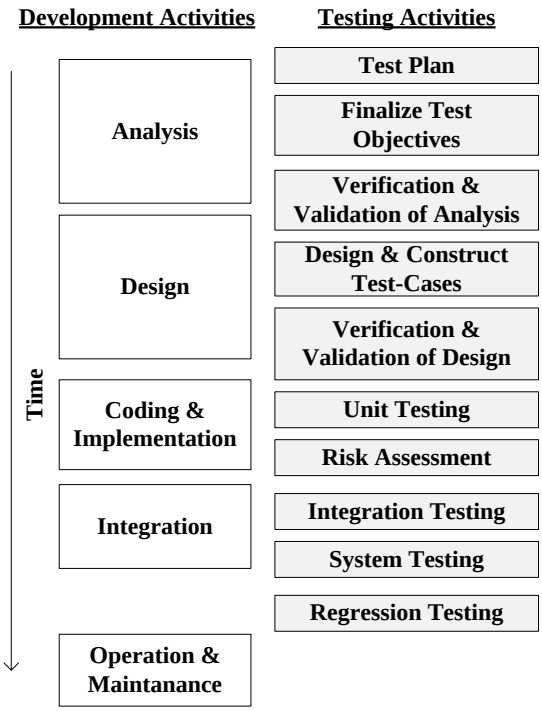

Fig 4: Testing activities in parallel with development activities.

3.1 Test Team Building

The purposes for building the test team are:

a) To perform formal testing at different stage of software testing methodology.
b) To perform verification and validation of the deliverables.
c) Act as consultants to development team in unit testing.

3.1.1 Key Application Areas Identification

Before starting the testing process, it is important to identify the key application areas. Testing is divided into the application areas by assigning testing group's responsibilities to those areas [10]. The statement of application areas come out as output from this step.

3.1.2 Find Out the Key Individuals

It is the test engineers who will perform all the tasks of testing methodology. So, the persons who are involved directly and indirectly in the testing process must be identified with care. The statement of team members come out as output from this step.

3.1.3 Build an Effective Test Team

Building a test team doesn't only mean a collection of group members but also taking the necessary steps to bring the desired outputs from them. Building an effective test team includes the following steps [11]:

a) Let the team members know the reason they have been selected for the team.

b) Assigning ownership to individuals.

c) Include knowledge of seasoned players in the team.

d) Motivate and recognize their accomplishments.

3.1.4 Responsibilities Assignment

Responsibilities must be assigned to team members at group level and individual level to finish the activities according to the schedule. The team work plan comes out as output from this step.

3.2 Risk Assessment

The purposes of risk analysis during software testing are:

a) Identify high-risk application modules to test them thoroughly.

b) Identify error-prone components within specific applications to test them precisely.

c) Reduce testing efforts and cost of the software.

3.2.1 Methods for Risk Analysis

To achieve the best result from risk analysis it must be done in a formal way. Otherwise it produces ineffective results.

There are three main *'risk analysis'* methods for software testing [12]:

a) *Judgment and Instinct:* This method is commonly used for performing risk assessment during testing. Using this approach the amount of testing required for a project can be estimated.

b) *Dollar Estimation:* This method is also known as a function of loss and event probability. This approach measures the business risks using dollars as its unit. The loss is calculated using the following equation.

$$\lambda = \rho \times \varphi$$

where, λ = Annual loss expectancy, ρ = Single loss expectancy, φ = Annual rate of occurrence.

c) *Identifying and Weighting Risk Attributes:* This method finds out the attributes for causing a risk. The relationship between the attributes and the risk is determined by weighting. The testers identify high risk areas by using weighted numerical scores. The scores can be used to decide which application components should be tested more thoroughly than others [10] [13].

3.2.2 Dimensions of Risk

The common three risk dimensions are:

a) *Structure of the Project:* When a project is well structured then the risk is less and vice versa.

b) *Size of the Project:* If a project size is small the risk is low and if the size is large then risk is high.

c) *Technological Experience:* High technology experience reduces the risk in a project and vice versa [14] [15].

3.2.3 Risk Analysis steps

The risk analysis process is carried out by the following steps:

a) *Risk Point Calculation:* For calculating the risk score, first a set of dependency factors is prepared to formalize the process as shown in Table 2. For a module each dependency factor is discussed in group session to the team members related to that specific module. These are closed-end questions with the possible responses of *Low*, *Medium*, *High*, and *Not Applicable*. Each question has numeric risk values according to its importance and response. A numeric rating is assigned to the response.

For deciding the area importance factor first risk analysis method i.e. *Judgment and Instinct* has been used. The final risk score for a module is calculated by multiplying the risk score by area importance factor as the following:

$$\pi = \alpha \times \upsilon$$

Here, π = final risk point, α = average risk point and υ = area importance factor. A total score is computed for the project, but the individual scores are used to develop the risk profile. The individual scores are calculated

Table 2. Risk Dependency Matrix

SL #	Dependency Factor	Options Available	Select Option	Risk Point
1	Formality of Development Process	*High=0, Medium=10, Low=20, NA=0*		
2	Developer's Experience	*High=0, Medium=10, Low=20, NA=0*		
3	Formality of Configuration Management Process	*High=0, Medium=10, Low=20, NA=0*		
4	Formality of the QA process	*High=0, Medium=10, Low=20, NA=0*		
5	Relationship with users	*High=0, Medium=9, Low=18, NA=0*		
6	Accessibility of the user documentation	*High=0, Medium=8, Low=15, NA=0*		
7	Formality of the Coding Process	*High=0, Medium=8, Low=15, NA=0*		
8	Formality of the Coding Documentation	*High=0, Medium=8, Low=15, NA=0*		
9	User Approvals	*High=0, Medium=8, Low=15, NA=0*		
10	Code Reviews	*High=0, Medium=8, Low=15, NA=0*		
11	Accessibility of the Development Standards	*High=0, Medium=5, Low=10, NA=0*		
12	Formality of Design Documentation	*High=0, Medium=5, Low=10, NA=0*		

13	Accessibility of User Documentation	*High=0, Medium=4, Low=8, NA=0*		
14	Accessibility of supporting Documentation	*High=0, Medium=4, Low=8, NA=0*		
15	General Merging for Error	*High=0, Medium=3, Low=6, NA=0*		
16	Accessibility of Visual Interface Standards	*High=0, Medium=3, Low=6, NA=0*		
17	Effectiveness of communication with users	*High=0, Medium=2, Low=4, NA=0*		
18	Dependence on Technical Contributions from other Areas	*High=0, Medium=2, Low=4, NA=0*		

b) *Resources Allocation:* The common strategy is to allocate the most test resources in the high-risk areas, less testing resources to medium risk areas and minimal testing resources to low-risk areas. A possible distribution of resource could be to allocate 80% of the testing resources to medium and high risk areas and the remaining 20% to low risk areas.

c) *Risk Assessment Repository Creation:* Maintaining a risk assessment repository has two significant purposes. First, this data can be used for future risk assessment process which improves the risk assessment process itself. Second, the data becomes helpful for management plan and structure development projects.

Pseudo code for risk point calculation and resource allocation

Begin

1. $D = \{d_1, d_2, d_3...., d_n\}$ a set of dependency factor
2. Identify development modules $M = \{m_1, m_2, ..., m_n\}$
3. For m_1 to m_n
   i. Determine *area importance factor, I*
   ii. Take a developer involved in consideration
   For $d_1$to d_n
   Calculate *risk point, P*
   End For
   Calculate *total point,* $P_t = SUM\ (P)$
   iii. If all developer considered Then
   Calculate average *risk point* as
   $P_A = SUM\ (P_t)\ /\ COUNT\ (Developers)$
   Else
   Go to Step II
   End If
   iv. Calculate *final risk point,* $P_F = P_A * I$
   End For
4. If P_F is *High* Then
   Allocate more resources
   Else
   Allocate less resource
   End If

End

3.3 Finalize Test Objectives

The purposes of finalizing the test objectives are:

a) To know what the tester wants to accomplish when implementing a specific testing activity.

b) To intuitively explain why we are executing a particular step in the testing process.

c) To see that test objectives works according to inputs while writing test cases.

d) To find that test objectives work as checklist for validation & verification of requirements.

3.3.1 Test Objectives Identification

Three methods can be used to specify test objectives as described below:

a) Test objectives are determined based on analysis of the documents and diagrams of the requirements specification. It should be performed as a walk-through of the specifications section by section. The test team uses brainstorming for deciding the objectives.

b) The second approach is to identify key system functions then specify test objectives for each function. This procedure can also be performed as a walk-through of the functional requirements section by section.

c) The third method is to identify business transactions and base objectives on them. This can also be thought of as scenario-based as business cycles could be used to drive the process.

3.3.2 Completion Criteria

Each test objective has one or more completion criteria. If these completion criteria are satisfied then it is considered that the test objective is finished. In this way completion criteria work as the standard for measuring test objectives. Completion criteria can be either quantitative or qualitative.

3.3.3 Prioritize Test Objectives

The test objectives should be prioritized based on the risk analysis findings. For testing the high and medium priority test objectives, more testing efforts are employed while minimum effort is given for low priority test objectives.

3.4 Construction of Test Plan

The test plan is an operational document. It can be used for both new development and maintenance. Most important steps in constructing test plans are:

a) Plan the Approach

b) Determine the Features to be tested

c) Design the Tests

d) Construct the Tests

e) Execute the Test Procedures

The purposes of constructing the test plans are

a) To specify who does what, and when and why of the test design, test construction, test execution, and test analysis process steps.

b) To define the test environment(s), including site locations, hardware and software components, test equipment, and personnel needed for the tests.

c) To identify the type of tests to be performed.

d) To provide measurable goals by which management can gauge testing.

e) To facilitate communications within the test team, between the test team and the development team, and between the test team and management.

f) To provide test schedules for each test [15] [16].

3.5 Construction of Test Cases Design

The purposes of writing test cases are:

a) To design and build a set of *"intelligent"* test data.

b) To validate the testing coverage of the application.

c) To bring some sort of standardization and minimizes the ad-hoc approach in testing.

3.5.1 Test case

A test case is a set of conditions or variables under which a tester will determine whether an application is working correctly or not. A test case has components that describe an input, action or event and an expected response, to determine if a feature of an application is working correctly.

3.5.2 Choosing Test Data

The test cases should test the system thoroughly as well as focus on high-risk areas and on system components where weaknesses are traditionally found.

It is not economically feasible to test every possible situation, so representative sampling of test data is required. In order to fully test that all the requirements of an application are met, there must be at least two test cases for each requirement: one positive test and one negative test; unless a requirement has sub-requirements. In that situation, each sub-requirement must have at least two test cases.

So, $T = 2 * N$

Where, T = minimum number of test cases, N = no of requirements.

3.5.3 Designing the test cases

This task involves applying the test case design techniques to identify test data values that will be constructed. The developers will be responsible for designing unit testing test cases. The test team will aid developers in use of the techniques. For integration, system, regression and progression testing the test team will apply the techniques themselves. The test case description can be documented either manually or stored in the test repository of an automated testing tool suite.

3.5.4 Construction of the test data

This task is the construction of the actual physical data sets that satisfy the test cases designed in the previous task. The medium in which the data are constructed is determined at the time of construction.

3.5.5 Construction of the Test Logs

A separate test log should be constructed for the unit test, for the integration test, for each of the system tests, and for all subsequent regression and progression tests. This log can be created with a spreadsheet and printed before testing when testing is implemented manually.

3.5.6 Test Case writing

Depending on the application type, organizational *QA* strategies the format of the test case can be different. Generally test cases should contain the following fields:

a) Test case *ID* or *No*

b) Unit or Module to be tested

c) Related Use Case Name

d) Test data

e) Description

f) Steps to be executed

g) Expected result

Pseudo code for writing test cases is given below:

Begin

1. Identify the set of requirements $R = \{r_1, r_2, r_3, \ldots, r_n\}$
2. For each r_i in R
   Determine testing scope and number of test cases, *n*
   For *1* to *n*
   a. Enter the *ID* of the requirements
   b. Create test case *ID*
   c. Write short description for the test case
   d. If the test is positive Then
      i. Generate test data for which the system will pass the test
      ii. Determine the expected positive Result
      Else
      i. Generate test data for which the system will fail the test
      ii. Determine the expected negative Result
      End If
   e. If user interface is related Then
      Write steps to be executed in the user interface
      Else
      Write steps for seating input data
      End If
   End For
   If end of requirements Then
   Stop
   Else
   Go to Step 2
   End If

End

3.6 Unit Testing

The purposes of unit testing are:

a) To prove that the software functions as individual units i.e. to show that each unit does what it should do.

b) The unit test motivates the code that the developers write. It works as a little design document that says, what that part of the code will do [17].

c) Unit tests improve the maintainability of any application. Bugs are found right after they get introduced which reduce the cost to fix them and re-factoring code can be done easily.

The software developers test the individual units of source code while writing them. The *QA* engineers work as consultants to the developers during unit testing [18] [19]. Unit tests are most likely be defined at the method level, so the art is to define the unit test on the methods that cannot be checked by inspection. Usually this is the case when the method involves a cluster of objects. Unit tests that isolate clusters of objects for testing are doubly useful, because they test for failures, and they also identify those segments of code that are related [10] [17] [18]. Unit testing is done using the following steps:

a) Approve Test Environment

b) Approve Test Resources

c) Execute Unit Tests

3.7 Integration Testing

The purposes of integration testing are:

a) To prove that the software units work together properly.

b) To prove that the integrated units do what they are intended to do, and that the integrated units do not do things they are not intended to do.

When two or more tested units are combined into a larger structure, they need to be tested again to conform that they working as an integrated structure as well. This testing is known as integration testing. The formal test team should perform the integration test. Each unit is tested before combining. So when errors discovered while combining them it is easy to come to a decision that errors are related to the interface between units. This approach reduces the effort for finding interface errors [19].

Integration testing can be done in a variety of ways but the following are three common strategies:

a) Top-down approach

b) Bottom-up approach

c) Umbrella approach

Integration testing steps are:

a) Approve Test Environment and Test Resources

b) Execute Integration Tests

c) Retest Problem Areas

3.8 System Testing

The purposes of the system test are:

a) To use the system in a *"controlled"* test environment, but to do so as the user would use the system in the production environment.

b) To prove that the complete system does what it is supposed to do and does not do anything that it is not supposed to do.

c) To test the system as a whole for ensuring that it meets the quality standards.

System testing is performed to ensure that the entire system is working according to its specified requirements. Before performing the system testing it will be ensured that all the integrated components of the software have successfully passed integration testing and software system itself also integrated with applicable hardware system.

System testing is performed by the *QA* engineers. System testing find defects within the inter-assemblages and also within the whole system [19].

Common three types of System Testing are:

a) System Verification Testing.

b) Customer Verification Testing and

c) Customer Validation Testing.

Steps involved in system testing are:

a) Developing System Test Cases

b) Establishing and Documenting the Expected Results

c) System Testing and Automated Testing Tools

d) Retest Problem Areas

3.9 Regression Testing

The purposes of regression testing are:

a) To prove that system enhancements, and routine tuning and maintenance do not affect the original functionality of the system.

b) To prove enhance or maintenance changes do what they are intended to do, and do not do anything they are not intended to do.

After a system passes the system testing when modification in any module or functionality is done the system needs to be tested again as whole. This is to ensure that new change does not introduce any kind of defect in any part of the system. This testing method is known as regression testing. Unexpected faults can be detected by regression testing. Especially defects that occur for lack of understanding the internal code correlations when modifying or extending code by developers. Every time regression testing should be used to check the code's integrity when code is modified or used in a new environment. Regression testing is completed using the following two steps:

a) Building a regression test library

b) Execute regression tests

Execution of regression tests is done in two phases:

a) Execute old test data

b) Execute new test data

4. EVALUATION

The improvements made by the proposed model have been discussed in this section. Software test data has been collected for existing software testing methodologies from four software development organizations in Dhaka, Bangladesh. Three software also have been tested using the proposed

testing model. In the following sections we analyze and validate the advantages of the proposed testing model.

4.1 Coverage of Development Activities

The proposed software testing methodology run in parallel with the software development activities from the requirement analysis to maintenance level. It ensures the verification and validation of development activities from the beginning to the end. Testing life cycle of the proposed model is compared with the existing models and shown in graphically in figure 5.

The proposed model ensures the maximum coverage for the development process. Therefore, faults are detected as soon as they are introduced. Specially, the proposed model introduces a clear and methodical approach for finding defects in the early phases of software development life cycle and thus reduces development cost of the software [1].

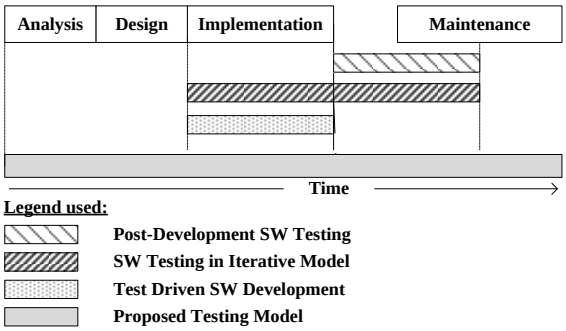

Fig 5: Coverage of development activities in different software testing methodologies

4.2 Development Effort Reduction

From collected data we filtered the effort for fixing the defects occurred during analysis and design phase of the development process. Here fault fixing effort with respect to total development effort was calculated using the following equation:

$$\text{Fault fixing effort} = \frac{\text{FaultFixingTime(ManDay)}}{\text{TotalDevelopmentTime(ManDay)}} * 100\ \%$$

Table 3. Fault fixing effort for analysis and design phase

SL#	Testing methodology	Fault fixing effort (%)
1	Average for Post-Implementation Software testing methodology	5.50%
2	Average for Modern Software testing methodology	3.70%
3	Average for Ad-hoc software testing methodology	11.45%
4	Average for Proposed software testing process model	1.13%

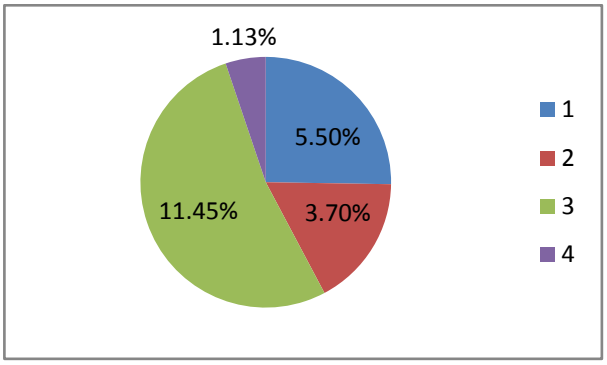

Figure 6: Pie chart of fault fixing efforts based on table 3.

From the analysis it has been found that the average fault fixing effort for existing software testing methodologies (methodology 1, 2 and 3 from table 3) is 6.88 %. On the other hand the proposed software testing model has the average fault fixing effort of 1.13%. Therefore using the proposed software testing model development effort can be reduced to 5.7% on an average.

4.3 Testing Effort Reduction

The risk assessment process in the proposed software testing model can reduce the testing effort significantly in system testing, regression testing and acceptance testing phases. Therefore the risk assessment phase reduces approximately 25% efforts for test execution part in system testing, regression testing and acceptance testing.

4.4 Early Project Release

Black-box testing must be done after the development of the system. However black-box testing effort is segregated into three parts *test plan*, *test-case design*, and *test execution*, where the first two parts are done in parallel with the development activities as shown in figure 7. Therefore only the test execution part of the black-box testing is done after the implementation.

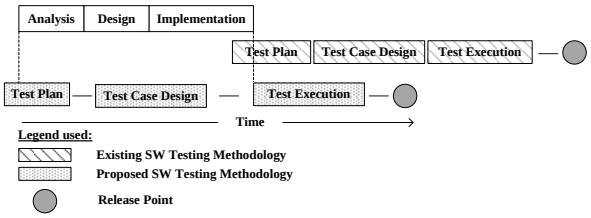

Fig 7: Black-box testing approach in existing and proposed testing methodologies

It has been shown that improvements have been achieved in three phases. First, verification and validation of analysis and design reduces the total development effort by 5.7%. Second, risk assessment phase reduces some percentage of the total project activities from the post-implementation part. Third, proposed black-box approach shifts some percentage of total project activities from the post-implementation part to the pre-implementation part. Therefore this approach ensures early project release by 15 – 20%.

In the proposed software testing process model success of verification and validation of analysis and design largely depends on determining proper test objectives. Therefore test-objectives must be finalized with proper steps and questionnaires must be selected with care. The cost for risk assessment is negligible with respect to its cost reduction in test execution in black-box testing.

5. CONCLUSION

In this paper, software testing process model was proposed that introduced testing at requirement analysis phase as well as incorporated risk minimization techniques. The proposed software testing model covered the whole software development process. So, there would be little chance of skipping any software defects and it ensured almost bug free quality products. In this testing model, tests run throughout all the different stages of the development life cycle. So, defects were detected as soon as they were introduced. Early defect detection or in another way the defect prevention reduced the cost and effort for fixing them. We verified our model with data from four different software development organizations. Also we applied the model for three different software, where we found gain in development effort by 5.7% on an average.

The proposed model also integrated risk assessment process with test management. It identifies and assigns scores to different kinds of risks based on their implications on project, thus diverting more resources and testing efforts to high risk areas. It eventually reduced testing effort in post-implementation phase.

This proposed test process model is a well defined software testing process. It does not depend on any particular test engineer. New engineers are able to continue the testing at ease and the testing schedule should be always on target.

6. REFERENCES

[1] Software errors cost U.S. economy $59.5 billion annually, NIST report. URL: http://www.nist.gov/director/planning/upload/report02-3.pdf

[2] McConnell, Steve (2004). Code Complete (2nd ed.), Microsoft Press, ISBN 0-7356-1967-0.

[3] Priyanka Chandani and Chetna Gupta, A survey on effective defect prevention - 3T approach, International Journal of Information Engineering and Electronic Business, 2014, vol. 1, pp. 32-41.

[4] Daniel Simon, and Frank Simon, Whitepaper / Integrating test management and risk management, October 2012.

[5] Chhavi Malhotra and Anuradha Chug, Agile Testing with Scrum-A Survey, International Journal of Advanced Research in Computer Science and Software Engineering, vol. 3(3), March 2013, pp. 452 – 459.

[6] R. L. Glas, Agile versus traditional, Cutter IT Journal, vol. 14, pp. 12-18, December 2001.

[7] I. Sommerville, Software Engineering. PEARSON Education (Singapore), ISBN: 81-297-0867-1, 2006.

[8] F. ALGAN, Test driven software development, Master's thesis, Izmir Insti-tute of Technology, June 2005.

[9] R. Wieringa, Combining static and dynamic modeling methods: A comparison of four methods, The Computer Journal, vol. 38, pp. 17-30, 1996.

[10] A. Nawaz and K. M. Malik, - Software testing process in agile development, Master's thesis, Blekinge Institute of Technology, June 2008.

[11] S. R. Bhat, How to build a successful QA team… URL: http://www.softwaretestinghelp.com/how-to-build-a-successful-qa-team/ [Last Visited: 2014-12-02].

[12] D. Verdon & G. McGraw, Risk analysis in software design. URL: http://www.cigital.com/papers/download/bsi3-risk.pdf [Last Visited: 2014-12-02].

[13] K. Whitmill, Risk-based testing, Journal of Software Testing Professionals, pp. 1-15, September 2001.

[14] S. Vernersson, Penetration testing in a web application environment, Master's thesis, Linneaus University, October 2010.

[15] C. Borysowich, Test plan development. http://it.toolbox.com/blogs/enterprise-solutions/testing-test-plan-development-step-1-2923 [visited: 2014-12-02].

[16] Y. Li and P. D, Software testing in a system development process: A life cycle perspective, Journal of Systems Management, pp. 23-31, 1990.

[17] T. Burns, Effective unit testing. URL: http://ubiquity.acm.org/article.cfm?id=358976 [Last Visited: 2014-12-02].

[18] H. Zhu, P. A. V. Hall, and J. H. R. May, Software unit test coverage and adequacy, ACM Computing Surveys, vol. 29, pp. 366-427, 1997.

[19] L. Luo and P. Li, Software testing techniques, 2001.