A Process Framework to Control the Time of Software Acceptance Testing

Ahmed El-abbassy
El-shorouk Academy, Institute of Computers and Information Technology
P.O Box, El-shorouk, Cairo, Egypt
ahmed_elabbassy@yahoo.com

Hamdy Riad
El-shorouk Academy, Institute of Computers and Information Technology
P.O Box, El-shorouk, Cairo, Egypt
hamdy_riad@yahoo.com

Abstract:

Acceptance testing is the formal testing phase used to demonstrate that the software performs as required. This is the final stage in the testing process before the software is accepted for operational use.

Acceptance testing is the responsibility of the customer. Nevertheless, it has important implications on the project, as its duration impacts the cost and the payment schedule.

It is therefore important for both customer and software supplier to ensure the thoroughness of the acceptance test while minimizing its duration.

The lack of coordination between acceptance testing and other testing phases (unit testing, integration testing, and system testing), in addition to customer's wish to build confidence in the software, lead to a substantial prolongation of acceptance testing duration.

In this paper, we discuss a practical approach to shorten the time of acceptance testing, and we propose a framework for "good enough" testing derived from the following basic ideas: Involving the customer early in the software testing activities and consider the scope and results of the other testing activities when planning for acceptance testing.

Combining acceptance testing with other testing activities will save time and money.

Keywords: Software Engineering, Software Testing, and Acceptance testing.

1. Introduction

Testing plays an important role in today's software development life cycle. During testing, we follow a systematic procedure to discover defects at various stages of the life cycle.

Software testing is the most costly activity in a software project; roughly 40% of the total cost is spent in software testing [1].

Software testing is a mission-critical quality assurance function. It is a systematic process planned and integrated with the software development process.

Software testing activities are normally grouped into several phases [1] as illustrated in figure (1):

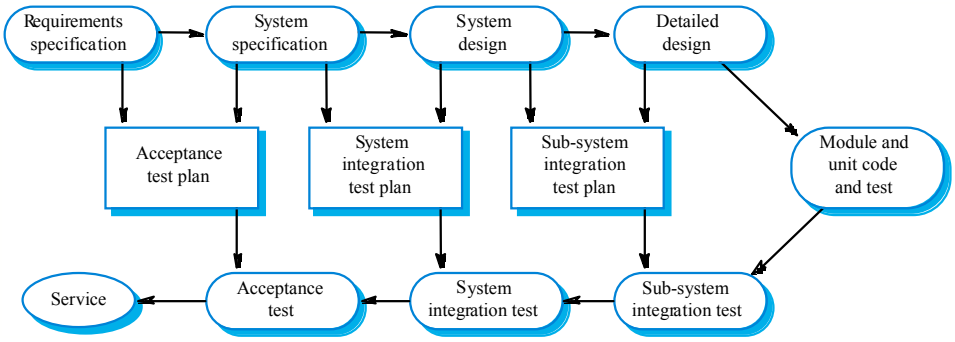

Figure (1): Software testing lifecycle

The objectives of the different test phases are:

- Unit testing: The unit testing focuses on testing the smallest unit of the system design, which is the module. It addresses the correctness of the functions incorporated in a single module and its dealing with error conditions.
- Integration testing: Integration testing involves testing the interfaces between collections of modules, which have been integrated into applications. The integration testing focuses on the detection of interface errors by rigorously exercising these interfaces.
- System testing: The aim of the system testing is to validate the system requirements. It provides final assurance that the system software meets all functional requirements. The system testing includes the application functionality, performance, and appropriateness, demonstrating that the system meets its requirements within the defined constraints (e.g., performance, security, recovery).
- Acceptance testing: Acceptance testing is the formal testing phase used to demonstrate that the software performs as required. This is the final stage in the testing process before the software is accepted for operational use.

- The acceptance testing focuses on finding flaws in how the system meets requirements, procedures, and usability from customer point of view.

2. Software acceptance testing

2.1 Objectives

Acceptance testing is a critical stage of the testing process since it is managed by the users to determine if the software meets the terms of the requirements document [2].

2.2 Scope

This test consists of a series of test cases, with defined expected results, that will validate the functionality of the system and ensure that the users can work with the system as it has been designed.

Acceptance testing, which is sometimes combined with a simulated work environment, should also ensure that the users can complete a day's work in a work day.

The primary characteristics evaluated during acceptance testing are:

- System functions and data.
- System security.
- Human factors.

Acceptance testing should demonstrate that the software meets the original business objectives, satisfies the user requirements, and operates within the constraints that were defined. Similarity between acceptance and system testing is also apparent in the individual tests that make up acceptance testing (usability tests, volume tests, stress tests, and so forth).

Unlike system testing, however, acceptance testing is the responsibility of the customer.

2.3. Acceptance test definition principles

The following principles for the definition of test cases apply to acceptance test strategy and design performed by the customer.

- Tests should be realistic as possible, preferably based on existing real data rather than with simulated test data to make the definition of the test set up, the test data and the test results easy and verifiable. Test data is prepared during the development phase to represent sample real data.
- Tests serve to verify that the system is performing according to the defined requirements and specifications, not to introduce additional features that were not originally in the scope.
- Test cases should be independent from each other. This allows the execution of the second case even if the first did not complete correctly and makes it easier to find the cause of erroneous result.

- The acceptance test suite is implemented in a way that allows the execution of a specific single test, of a set of tests and of the complete suite. Automatic test set-up, execution and result evaluation is of great advantage for regression testing (e.g. when checking incident resolutions or testing an upgrade) and should be aimed for.
- Acceptance testing is conducted using hardware and software of a test platform and/or the operational environment.

2.4 Management of acceptance testing

- The formal acceptance test is conducted based on acceptance test cases and acceptance test plan.
- Achieving Acceptance: When all test cases defined for the acceptance test of a specific deliverable have been executed and their results have been recorded a test report is compiled.
- The acceptance testing is considered to be completed upon satisfactory performance of test cases and resolution of the major discrepancies of system requirements.

3. Definition of the problem

Acceptance testing is a formal testing phase with an objective to give confidence to customers on the software subject to delivery.

The acceptance testing is solely the responsibility of the customer [2]. Nevertheless, it has important implications on the project, as the duration of the acceptance testing impacts the cost and the payment schedule [3].

It is therefore important to the customer and the software supplier to ensure the thoroughness of acceptance tests while minimizing their duration.

However, acceptance testing is preceded by an extensive testing effort organized in three test phases:

- Unit testing
- Integration testing
- Validation testing

These phases are considered by the software supplier as internal processes and their results are quasi invisible to the customers [4].

In addition, there is always a lack of coordination between acceptance testing and other testing phases, and it is clear that there are inevitably overlaps between the tests performed in the different test phases.

According to practice, acceptance testing process degenerates, and never completed in time.

Through acceptance testing, the customer tries to build confidence in the software by **testing everything again**. This means a dramatic increase in acceptance testing time.

4. A proposed process framework to control acceptance testing time

A process framework is a skeleton that specifies and coordinates the various processes necessary to complete a complex task.

The aim of the proposed framework is to set principles and rules that provide the basis for solving the complex issue of how to perform effective acceptance testing in the project while minimizing its duration.

The proposed approach to improve acceptance testing time is derived from two basic ideas:

- Involving the customer early in the software testing activities.
- Considering the scope and results of the other test phases when planning for acceptance testing.

By Combining and coordinating acceptance testing with other testing activities, we can save time and money.

Coordinating testing activities shall positively contribute to:

- Deliver the software faster
- Improve the user trust in software

The skeleton of the proposed testing process is illustrated in Figure (2)

The Skeleton of the Testing Process

Start
1. Prepare and sign-off test plans and detailed test procedures
2. Prepare test data for the different test phases
3. **for** each test phase (including acceptance testing)
4. **do** Run tests with client involvement
5. Compare results to test cases and calculate quality figures
6. **while** (Actual quality Figures are unsatisfactory)
7. **do** Perform corrective actions and update quality figures
8. Sign-off test phase
9. Perform additional AD-Hoc testing by the client(*)

End

(*) AD-Hoc testing is an optional activity where the client may run a set of random tests in order to be more confident in the software before staring the operational use. The duration of this step should be fixed and agreed upon by both the client and the software supplier.

Figure (2) the overall testing process

As illustrated in Figure (2) the basic element of the process model is the activity. An activity is defined to accomplish a specific task (group of related tasks).
Each activity has entry conditions, exit conditions and responsibilities.

Specification	Description
Entry	Conditions to be met before task initiation
Exit	Results produced
Responsibilities	The role of the software supplier and the client

Activity #1: Prepare and sign-off test plans and detailed test procedures.

Specification	Description
Entry	- Inspected and approved: Requirements, Design, Code and changes.
Exit	- Inspected and approved test plans and procedures. - Traceability between specifications and test cases. - Number of tests per phase. - Metrics used to assess the quality software testing. - End of test phase fail/pass criteria.
Responsibilities	- This task is the responsibility of the software supplier. - This step shall be approved by the client.

Activity #2: Prepare test data.

Specification	Description
Entry	- Inspected and approved test plans and procedures.
Exit	- Sample real and simulated data used for testing.
Responsibilities	- This task is a joint responsibility of the software supplier and the client.

Activity #3: Run tests with client involvement.

Specification	Description
Entry	- Inspected and approved test plans and procedures. - End of test phase fail/pass criteria.
Exit	- Test results recorded using test forms.
Responsibilities	- This task is the responsibility of the software supplier. - Client shall be a witness, and shall sign the test result forms.

Activity #4: Compare results to test cases and calculate quality figures.

Specification	Description
Entry	- Inspected and approved test plans and procedures. - End of test phase fail/pass criteria. - Metrics to assess the quality of software testing.
Exit	- The problems detected during software testing. - Statistics on test results and measures of testing quality.
Responsibilities	- This activity is the responsibility of the software supplier. - Results shall be made visible to client. - Client shall approve the test result forms.

Activity #5: Perform corrective actions and update quality figures.

Specification	Description
Entry	- The problems detected during software testing.
Exit	- Test results recorded using test forms - Updated Statistics on test results and measures of testing quality.
Responsibilities	- This activity is the responsibility of the software supplier. - Results shall be made visible to client. - Client shall approve the test result forms.

Activity #6: Perform additional AD-Hoc testing by the client.

Specification	Description
Entry	- Test cases defined by the client - Metrics to assess the quality of software testing.
Exit	- The problems detected during software testing. - Statistics on test results and measures of testing quality.
Responsibilities	- This activity is the responsibility of the client. - Software supplier shall be a witness. - Client and software supplier shall approve the results.

The proposed approach is implemented using widely accepted software engineering concepts and techniques which are introduced in the following sections:

- Traceability
- Visibility of software testing process.
- Testability of each requirement.
- The use of metrics to assess the quality of software testing**.**

4.1. Traceability [5, 6, and 7]

Requirements traceability is defined as the ability to describe and follow the life of a requirement, in both forward and backward direction.

In the Requirements management process area, specific practice states, "Maintain bidirectional traceability among the requirements and the project plans and work products (i.e., from requirements to end products and from end product back to requirements).

Such bidirectional traceability helps to determine that all source requirements have been completely addressed and that all lower level requirements can be traced to a valid source.

Traceability between functional requirements and test cases should be maintained.

Traceability can be achieved by using many techniques; the most popular technique is traceability matrix.

A traceability matrix is a verification tool to trace a requirement throughout the lifecycle. It should be developed because it provides visibility into completeness of the quantitative definition and testability of each requirement.

The following table (1) shows an example of the relationships between the Software Requirements Specification (SRS) and the Software Test Plan (TST). In this case, Software Test Plan Requirement 1 (TST 1) is traced from Software Requirements Specification Requirement 1 (SRS 1) while TST 2 is traced from SRS 2 and SRS 3.

Table (1): a traceability matrix table

	TST 1	TST 2
SRS 1	X	
SRS 2		X
SRS 3		X

4.2. Testability of each requirement [8, 9]

In order to minimize the time allocated for acceptance testing, planning for software testing should have the following two main objectives:

- Ensuring the testability of each requirement.
- Distributing the test cases among the different test phases, in order to avoid repeating same tests many times.

Software engineering techniques, to achieve such objectives are:

- Applying conveniently the functional testing techniques such as boundary value analysis and equivalence class partitioning.
- The use of regression testing.
- The use of sample real data during software testing rather than with simulated test data.
- Predicting the number of test cases required in order to test adequately the developed software.

4.3. Visibility of software testing process [10]

Visibility means that the software process activities culminate in clear results so that the progress of the process is externally visible.

To help the customers to maintain control, the supplier should provide a transparent application development infrastructure.

To achieve visibility regarding software testing process, the following provisions should take place:

4.3.1. Test logs and reports

The results of executing the tests comprised in the different test phases should be documented in test logs and test reports.

The test logs shall document the execution of the tests. It provides a record of relevant details about the execution of tests. For this purpose, a test checklist as illustrated in Table (2) may be used. The execution of each round of testing will lead to an update of the test checklist, and for each executed test, the test result is noted (test successful or not). For a failed test, an incident report shall be issued and its number will be recorded in the test checklist. As an example, an incident report form is illustrated in Table (3).

The test report summarize the test checklists for any given test event at specific points in time for the purpose of reporting status. It will show the testing event's progress, issues and recommendations and provides an overview of the incident reports generated by the testing activity.

Table (2): Test Checklist Form

Test Checklist			
Completed By:	Effective Report Date:		
System Name:	Version No:	Test Event:	
Reviewed By:	Review Date:		
Test Case Id	Test Date	Actual Results	Incident Log ID

Table (3): Incident Report Form

System Name	Prepared By	Date Prepared	Incident Category	Incident Report No.
Incident Description				
Title				
Incident Definition				
Appended Information (E.G. Configuration File,…)				
Incident Resolution				
Incident Status	Date	Responsible	Remarks/Intermediate Corrections	Final Correction

4.3.2. Incident categories

The problems detected during software testing should be categorized to reflect their impact on the usability of the system, and their severity level.

Incident severity is a measure of the consequence of the defect.

An example of such classification is given in table (4)

Table (4): Incident Categories

Severity level	Incident Consequences
1(Critical)	A serious problem that makes further testing/work impossible until it is resolved.
2(Serious)	A significant error prevents the system from being operational but other tests/work can continue.
3(Medium)	An operational workaround has been provided for an error that would prevent the system being operational.
4(Low)	A minor issue that does not prevent the system from being operational but may inconvenience the users.

4.3.3 Managing software testing activities

The management requirements for the different test phases comprise:

- A test plan should be developed.
- Test results should be recorded and approved.
- Measures of the test results including for example:

  - The total number of tests.
  - The number of failed tests.
  - The number of successful tests.

Should be collected and analyzed, and confined to the customer.

- An agreement should be settled regarding release management. some possible release management alternatives are:

  - All known problems are resolved prior to release of the software.
  - All known problems of Severity Levels 1, 2 and 3 are resolved prior to releasing the software.
  - All known severity levels 1 and 2 failures are resolved prior to release of the software.
  - All known severity level 1 failures are resolved prior to release of the software.

4.4. The use of metrics to assess the quality software testing [11, 12, and 13]

The continuous monitoring of the testing process allows the establishment of an adequate level of confidence for the release of software products and for the quantification of software risks.

Monitoring of software testing process requires practical measurements for the quantification of all software testing phases.

Software metrics can help improve the testing process by providing insight and early visibility into the “real” status of the testing effort and helping to make assessments as to whether progress, productivity and quality goals are being met.

Defect Distribution, Defect Density and Defect Type metrics allow the quantification of the quality of the testing process.

Examples of such metrics are given in table (5)

Table (5): examples of software metrics

Test metric	Definition	Purpose	How to calculate
Number of defects	The total number of defects found in a given test phase that resulted in software modifications.	A meaningful way of assessing the stability and reliability of the software	Count the number of failed tests
Defect severity	The severity level of a defect indicates the potential business impact for the end user	Provides indications about the quality of the product under test. High-severity defects mean low product quality, and vice versa. At the end of this phase, this information is useful to make the release decision based on the number of defects and their severity levels.	Every defect has severity levels attached to it. Broadly, these are Critical, Serious, Medium and Low.
Defect Density	The number of defects per 1,000 lines of code. Or number of defects divided by the total number of tests	This metric indicates the quality of the product under test. It can provide an indication concerning the readiness for the software code to be released.	Ratio of the number of defects found vs. the total number of lines of code (thousands) or the total number of tests
Defect severity index	An index representing the average of the severity of the defects.	Provides a direct measurement of the quality of the product—specifically, reliability, fault tolerance and stability.	Two measures are required to compute the defect severity index. A number is assigned against each severity level: 4 (Critical), 3 (Serious), 2 (Medium), 1 (Low). Multiply each defect by its severity level number and add the totals; divide this by the total number of defects to determine the defect severity index.

5. Integrating the proposed framework with the software life cycle.

Table 6 below, illustrates how the proposed framework is integrated with the software development life cycle.

Table (6): Integration of the proposed framework with the software life cycle

Custom development life cycle phase	Proposed actions
Analysis	- The project shall perform and maintain bi-directional traceability between high level user requirements and the software requirements.
Design & Implementation	- The project shall perform and maintain bi-directional traceability between the software requirements and the software design - The project shall provide and maintain traceability from software design to the software code. - The project shall provide and maintain: a. Software Test Plan(s). b. Software Test Procedures. - The project shall ensure that test plans and procedures cover the product's complete functionality - The project shall distribute test cases among the test phases, in order to reduce repeating same tests many times. - The project shall provide and maintain traceability from the Software Test Procedures to the software requirements.
Software testing	- The project shall perform test cases as defined in the software test procedures. - The project shall provide visibility of the testing process to the customer. - The project shall provide and maintain a- Software Test Logs b. Software Test Reports. c- The problems detected during software testing shall be categorized to reflect their severity level. - The project shall provide and maintain a. Statistics on the test results b. Quantitative process data to measure the quality of the software product and the testing process. - The project shall not proceed to the next testing phase before achieving the required quality level in terms of the agreed upon defect density and / or defect severity index

6. Conclusions

This paper illustrates the following software engineering concepts:

- Traceability,
- Visibility,
- Software metrics
- Coordinating the activities of the different testing phases

These concepts can be translated into a set of concrete actions along the software life cycle that helps improve the testing process by providing insight and early visibility into the "real" status of the testing effort and helps to make assessments as to whether progress, productivity and quality goals are being met. We believe that this proposed framework shall contribute in shortening the acceptance testing time.

We expect to extend this approach to define more detailed relationships between these concepts and the acceptance testing attributes such as time, number of tests, and closure criteria.

References

[1] Ian Sommerville, Software Engineering Seventh Edition chapter 1, Pearson Education Limited, 2004.

[2] Tom Mochal, Acceptance testing: The customer is the ultimate judge, Builder.com, Published: 10/22/2001

[3] LESSONS LEARNED -- CURRENT PROBLEMS ,SPMN Software Development Bulletin #3, Copyright 2005 Integrated Computer Engineering, Inc.

[4] Software Quality Assurance-Support of Formal Software Testing, S&MA,QD-QE-012 REVISION C, Effective Date: September 24, 2004

[5] Requirements Tracing--An Overview, Liz Kean, Air Force Rome Laboratory, Copyright 2005 by Carnegie Mellon University

[6] Office of the Chief Engineer, NASA Software Engineering Requirements, NPR 7150.2, Effective Date: September 27, 2004

[7] Linda Westfall, Bidirectional Requirements Traceability, The Westfall Team, Copyright 2006

[8] Bach, J.S. (1997a) "Good enough testing for good enough software." Proceedings of STAR 97 (Sixth International Conference on Software Testing, Analysis, and Review, San Jose, CA., May 7, 1997, p. 659.

[9] BJ Rollison, Testing Techniques: Theory and Application. Software Test & Performance Conference, November 2006, Boston.

[10] Rob Pirozzi, LogiGear Corporation, Addressing Five Core Questions Surrounding Software Testing, LogiGear Corporation, Copyright 2006.

[11] Alfred Sorkowitz, Using Metrics to Improve Software Testing. Software Test & Performance Conference, November 2006, Boston.

[12] Kalyana Rao Konda, Measuring Defect Removal Accurately, software test & performance, by July 2005, p. 35.

[13] Stelios PANTELOPOULOS, SINGULAR S.A., Practical Measurements for Reengineering the Software Testing Process, EuroSPI 2000.