Software Quality Journal **6**, (1997) 137–149

A study of user acceptance tests

HARETON K.N. LEUNG and PETER W.L. WONG

Department of Computing, The Hong Kong Polytechnic University, Hung Hom, Hong Kong

Received January 1997

The user acceptance test (UAT) is the final stage of testing in application software development. When testing results meet the acceptance criteria, the software system can be released for operational use. This paper first compares the different testing phases of software development (i.e. unit test, integration test, system test and UAT) in terms of important testing elements so as to highlight the uniqueness of UAT relative to the other test phases. Then, we describe several approaches for acceptance test, including the behaviour-based approach, the black-box strategy and a new UAT strategy called *operation-based testing*. The new strategy uses the operational profile for testing purposes, includes a well-defined acceptance criteria, and satisfies the test requirements of ISO9001 standards.

Keywords: testing, reliability, user acceptance, acceptance criteria, operation-based

1. Introduction

Producing a high quality software system is one of the key objectives of software development. According to ISO9126, software quality can be evaluated by the following characteristics: functionality, reliability, usability, efficiency, maintainability, and portability. A study categorized the quality-carrying properties of software as: correctness, structuredness, modularity, and descriptive properties [1]. Throughout the development cycle, different software testing phases can be used to verify conformance to these quality-carrying properties.

Hetzel provides a complete guide for testing at different stages of the development cycle [2]. Hsia *et al*., [3] make use of the FSM models in the behaviour-based approach to generate the test cases based on scenarios. A new testing strategy making use of the operational profile has been proposed [4]. This concept is further extended in structural analysis to improve testing efficiency for system test [5].

Nevertheless, most of the existing test strategies are created for the developer rather than the user. The consequences are that some of the critical issues, such as operational procedures and operating environment, are rarely carefully considered during the course of testing. Since the software user provides the ultimate definition of software quality, a test strategy from the user's point of view is needed.

Software testing consists of the following phases: unit test, integration test, system test, and user acceptance test (UAT). UAT is the final stage of testing in application software development. When testing results meet the acceptance criteria, the software system can be released for operational use. In this paper, we propose a new UAT strategy. This strategy makes use of the previously established technology; incorporating the operational profile for testing and the use of ISO9000 standards for software development. It is proposed that the procedures for software acceptance be performed according to the test plan. Our strategy requires the establishment of test planning documentation (test plans, test procedures, test case design) before starting UAT, the formal user sign-off on the test results, and procedures for handling failed test cases. In particular, our strategy also establishes a well-defined acceptance criteria.

In Section 2, we compare UAT with other test phases including unit test, integration test, and system test in terms of several important testing elements. This highlights the differences between UAT and other tests. Section 3 first presents the behaviour-based approach and the black-box strategy for UAT. Section 3.3 then describes our proposed strategy. Finally, Section 4 compares and evaluates the three UAT strategies. Concluding remarks are given in Section 5.

2. Special characteristics of UAT

UAT tries to demonstrate the *start-to-end* functional capabilities of the software system and serves as a final confidence-building activity for the user [6]. The main objective is to evaluate the system readiness for operational use. Typically, the user selects the acceptance test cases. The UAT *acceptance criteria* are used to determine whether the software system is ready for deployment and should be jointly agreed upon by the developer and user. *Acceptance criteria* are the criteria that a system or component must satisfy in order to be accepted by the user, customer, or other authorized entity [7]. Once the acceptance criteria are fulfilled, the software system can be released for operational use.

To highlight the uniqueness of UAT, this section compares UAT with other test phases against the following test elements: test objectives, test criteria, test strategies, test oracles, test tools and environment, and tester.

Test objective or *focus* is an identified set of software features or components to be evaluated under specified conditions by comparing the actual behaviour with the specified behaviour as described in the software requirements [7]. Unit test is the lowest level of testing to check the functional correctness of the basic units or modules of the software application. The tester mainly tests the features and states of the software units, the handling of valid and invalid input, data structure, and unit boundaries. When software units are assembled into working groups, integration tests can be used to verify correct inferfacing. During the integration test, the tester exercises all modules and calls, program options and special utility routines, and even tries to *break* the integrated software skeleton. After the integration test, the entire system is tested as a whole. The focus of system testing is on the functional capabilities in handling valid and invalid inputs, and non-functional attributes, such as performance, reliability, and security. After system test, the software deliverable is handed over to the user for the user acceptance test to evaluate the readiness of the deliverable for operational use. In UAT, key test areas cover the major functions, user interface, and capabilities in handling invalid input and exceptions in operation.

Test criteria are used to determine the correctness of the software component under test. The test criteria of unit test is the extent of software unit free from faults in specification, design and coding. In integration test, the interfaces are tested to ensure they work according to the design. System test examines if the system as a whole functions and performs according to the specified requirements. The test criteria of UAT is the degree to which the software product and its documentation meet the acceptance criteria.

Test strategies are the methods for generating test cases based on formally or informally defined criteria of *test completeness* [8]. In unit tests, both white-box and black-box strategies are typically applied. The program units are usually tested against function and code coverage requirements. The software units are then integrated according to the hierarchy of the system design so as to build up a complete skeleton, for instance, on an incremental basis using a top-down or bottom-up approach. The big-bang approach may also be used to integrate all units together at the same time. Integration test cases are designed to test the unit interfaces and the interactions among the units. For system test, the tester checks for defects in the functionalities of the system and typically uses black-box testing. In UAT, the tester also uses black-box test cases to demonstrate the software readiness for operational use.

A *test oracle* is any program, process, or body of data that specifies the expected outcome of a set of tests [9]. In this paper, the *test oracle* refers to the *documentation* stating or implying what the expected output will be for a specified input. In unit test, the design specification serves as the test oracle. In integration and system tests, the functional and system specifications are the test oracles. User requirements, operational procedures, and quality manual are the test oracles for UAT.

Each test phase uses a different set of *test tools and environment*. Documentation tools are generally used in all test phases, since test documentation plays an important role in testing. Unit test is conducted with test drivers on the development machine to execute the units and with the use of code-coverage tools to ensure certain program components such as statements and branches are exercised. Tools such as a test case library, simulator and comparator are useful for integration and system tests. Testing is also carried out on the development machine. Test case and data generators are useful tools for all unit, integration, and system tests. In order to assimilate the use of a software deliverable in the operational environment, the UAT tester normally executes test cases on the operational platform (or in an operations-simulating environment).

In unit tests, the *tester* is normally the programmer or developer who develops the units. During integration and system tests, an independent test team may be formed. In UAT, the software deliverable is usually tested by the user. But, quite often, the user performs UAT with the developer who has the knowledge and experience to provide technical advice on test case selection.

After retrofitting work has been completed, the failed test cases must be rerun during the *regression test*. The regression test is the testing of the software application following amendment of software components, using existing test cases previously developed to test the original functionality of the software components [10]. There are no major differences among the different test phases. The concept of firewall proposed in [11] may be used to reduce the regression effort for integration testing.

Table 1 summarizes the differences and similarities of the four test phases.

3. Strategies for UAT

This section describes three UAT strategies. The first one is the *behaviour-based* approach proposed by Hsia *et al*. [3]. The black-box testing approach commonly used in the industry is then described. Finally, a new UAT approach is presented in Section 3.3.

3.1 *Behaviour-based acceptance test*

The *behaviour-based* (or scenario-based) *test* approach [3] is a recently proposed approach for acceptance testing by examining the external behaviour of the software system. Test cases are designed to test the system from the user's perspective by focusing on the external behaviour of the system. The test model consists of three submodels: the *user view, external system interface* and *external system view*. The procedures of the scenario-based acceptance test are:

1. elicit and specify scenarios;
2. identify the user's and external system views according to requirements;
3. formalize test model;
4. formalize the scenario trees and construct the scenario-based finite state machines (FSMs) which are then combined to form a composite FSM (CFSM);
5. verify scenarios;
6. verify the generated FSMs and CFSMs against their man–machine and external system interface.

Table 1. Comparing UAT and other test phases

Test elements	*Unit test*	*Integration test*	*System test*	*User acceptance test*
Test objectives	Check functions of each unit. Test unit's features and states, handling invalid input, data structure, boundary value	Test assembled units as working group. Test all modules and calls, and the interface	Test the functional and non-functional performance and capabilities of entire system	A demonstrative test to evaluate the system's readiness for use. Test major functions and interface, and common exceptions
Test criteria	Software unit free from design and coding defects	Same as unit test + software meets specified requirements	The extent of software meets specified requirements	Same as system test + software meets user expectations, i.e. acceptance criteria
Test strategies	White-box and black-box test techniques	Integrate modules using top-down, bottom-up, or big-bang strategy	Black-box test strategy	Black-box test strategy
Test oracles	Design document	Functional requirements specification	System and functional requirements specification	User requirements, operational procedures, quality manual
Test tools and environment	Code-coverage tool, documentation tool, test generator, test driver; test on development machine	Test generator, simulator, comparator, documentation tool; test on development machine	Test generator, test library, simulator, comparator, stress test tool, documentation tool; test on operation-simulating environment	Comparator, documentation tool; test on operational platform
Tester	Normally the developer	Developer or test team	Test team	User and sometimes with developer

After scenarios are specified, the scenario schema can be modelled so that the test scenario can be analysed, verified, and generated systematically. One of the weaknesses of this approach is that it can be time-consuming. Furthermore, the approach of testing the major system functionalities may be more appropriate for system test rather than for UAT. Although the submodel of user view is user-oriented, the external system interface and external system view are not user-oriented, as they require knowledge of system design. Only the developer can develop test cases from these views since he possesses the system design knowledge. Also, as the use of the FSM model emphasizes sequential processes, concurrent thinking in test design is difficult to put into practice [13]. The scenario approach covers only the normal test cases; the exceptions in operations are not covered. In addition, this approach does not specify the acceptance criteria from the user's perspective.

3.2 *Black-box approach for user acceptance test*

The black-box test strategy is deployed for UAT in most application software development projects. In this approach, the source of test cases is the functional or external requirements specification of the software system [2]. Test cases are often developed jointly by the user and developer. The user may execute test cases but will usually request the developer to provide the test plan and test input. A *functional test matrix* may be used to select a minimal set of functional test cases that *cover* the system functions. This matrix can provide systematic planning and coverage of normal cases so as to obtain a complete functional test set. The steps to select the requirements-based test cases are summarized as follows [2]:

1. Start with normal test cases from the specification and an identification of system functions.
2. Create a minimal set that exercises all inputs/outputs and covers the list of system functions.
3. Decompose any compound or multiple condition cases into single-condition cases.
4. Examine the combinations of single-condition cases and add test cases for any plausible functional combinations.
5. Reduce the list by deleting any test cases *dominated* by others already in the set.

The tester is typically knowledgeable in the application domain to include operational procedures in the testing. With the user's direct involvement, the tester may concentrate on procedural test cases and testing the handling of exceptional cases. An advantage of this test strategy is that it can reduce the possibility of having untested operational procedures. Apart from the balance of normal and exceptional test cases, another difficulty with this strategy is that the acceptance criteria are generally not clearly defined or easily determined by the user. The success of this strategy may rely on whether the developer can play the leading role in project planning and test management.

3.3 *Operation-based test strategy*

This section presents the framework of a new strategy for user acceptance testing. The *operation-based test (OBT) strategy* consists of the following key components:

1. test selection based on the operational profile;
2. well-defined acceptance criteria;
3. compliant with ISO9001 requirements.

We next describe the key components of the OBT strategy.

3.3.1 *Testing based on operational profile*

An empirical study by Ehrlich *et al* [16] demonstrates that the operational profile is important for pre-deployment testing of large-scale industrial software systems. An *operation* is a major task the system performs [18]. An *operational profile* for a system can be defined as a set of operations and their associated frequencies of occurrence in the expected field environment [12]. The OBT strategy incorporates the operational profile for testing purposes.

Developing an operational profile involves analysing the system usage and following five steps [12]:

1. Find the *customer profile.*
2. Establish the *user-profile*.
3. Define the *system-mode profile*.
4. Determine the *functional profile*.
5. Determine the *operational profile* itself.

Note that the operational profile is dependent on the users of the application. Most applications have several targeted user classes and each user class will have its own operational profile. Table 2 gives an example with three classes of users and four key operations. The occurrence probability of the same operation may be different for different user classes. During UAT, the reliability of the application must be checked against each of these operational profiles to ensure that the application is acceptable to all user classes.

Table 2. Operation profiles example

	Probability of use (%)			
Operational profile	*Operation 1*	*Operation 2*	*Operation 3*	*Operation 4*
1	20	20	30	30
2	50	10	10	30
3	10	40	25	25

In using the operational profile to drive testing, test cases should be selected according to:

1. The occurrence probabilities (relative frequency of use) of the operation. The amount of testing of an operation is based on its relative frequency of use. The most recently used operations will receive the most testing and less frequently used operations will receive less testing.
2. The *criticality* which measures the severity of the effect when the system fails. The more critical operations should receive more testing.

The following steps can be used to generate the test cases:

1. Select the operations for testing in accordance to their occurrence probabilities.
2. Select the *run category*. A *run* is a segment of program's execution, and a *run category* is a group of runs that exhibit homogeneous failure behaviour.
3. Identify and pick the input state that characterizes the specific run type. An input state uniquely determines the instructions a run will execute and determines the value of their operands. Test cases are selected randomly within each user operational profile.

3.3.2 *Test selection according to criticality*

System operations may be classified according to their criticality. We assume that the criticality of each operational profile can be determined separately. For example, a banking application may have three user classes: teller, supervisor, and manager. The teller class may be the most critical, the supervisor is less critical, and the manager the least critical in terms of the importance of the results of the application. According to the OBT strategy, the numbers of test cases for each operational profile should be proportional to its *weighted criticality*. The weighted criticality C_i' for an operational profile i can be computed as follows:

$$C_i^{'} = \frac{C_i}{\sum_{i=l}^{k} C_i P_i} P_i$$

where C_i is the criticality of operational profile i, P_i is its occurrence probability, and k is the total number of operational profiles.

Table 3 illustrates the calculation of the weighted criticality for a hypothetical case and demonstrates the test case selection strategy. Assume there are four operational profiles. Column

2 gives the criticality of each class of operational profile. Higher numbers indicate higher criticality. Column 3 lists the occurrence probability. Column 5 shows the computed weighted criticality. The final column identifies the number of test cases for each operational profile, based on the assumption that a total of 50 test cases will be used for testing the software system.

The following methods can be used to reduce the test effort [18].

1. Reduce the redundant test cases across different operational profiles and within the same operational profile. A test case may be used to test several operational profiles. It is more economical to store the test results and reuse them rather than running the same test cases for different profiles.
2. Reduce the numbers of operations and increase breadth of each operation.
3. Ignore infrequent operations.

Table 3. Example of test selection based on criticality

1	*2*	*3*	*4*	*5*	*6*
Operational profile	*Criticality (C_i)*	*Occurrence probability P_i (%)*	*C_iP_i*	*Weighted criticality (C_i')*	*Test cases*
1	3	30	0.9	0.391	20
2	2	40	0.8	0.348	17
3	4	10	0.4	0.174	9
4	1	20	0.2	0.087	4
Σ		100	2.3		50

3.3.3 *Acceptance criteria and acceptance decisions*

The OBT strategy uses the following acceptance criteria:

1. no critical faults detected, and
2. the software reliability is at an acceptance level.

The first acceptance criterion can be defined formally as follows:

Let T be a set of acceptance test cases and $t_i \in T$ with the mapping $s_i\ P\ s'_i$, where s_i is the input of test t_i; and s'_i is the output of applying s_i to program P.
The requirement of no critical fault implies that

$$\neg\exists s'_i \in C$$

where C is the set of incorrect critical output values as defined by the user.

If this criterion is fulfilled, the user may then apply the second criterion to check if the desirable reliability requirements are fulfilled.

The second acceptance criterion requires that the reliability of the software system due to non-critical faults is at an acceptable level. The software reliability is acceptable when the following two conditions are satisfied:

(a) The reliability of every operational profile class is acceptable when:

$$R_i \geq \Re_i \quad \forall i$$

where R_i is the estimated reliability of the operational profile i, and $\Re_i$ is the acceptable reliability of the operational profile i.

(b) The reliability of the whole application is acceptable when:

$$R_0 \geq \Re_0$$

where R_0 is the estimated reliability of the whole application, and $\Re_0$ is the acceptable reliability of the whole application.

$\Re_i$, $\Re_0$ are the pre-defined threshold limits for acceptance. They are set as the minimum reliability that the software deliverable should possess. Note that R_i, $\Re_i$, R_0, $\Re_0$ should be determined based on non-critical faults. In general, the acceptable reliability for the whole application should not be lower than that of individual operational profiles (i.e. $\Re_0 \geq \Re_i$). This is to ensure that the software deliverable has high reliability and integrity.

Our strategy makes use of the reliability model of [14] to estimate the reliability. The reliability of operation profile *i* can be estimated as follows:

$$R_i = 1 - [Nf_i / N_i]$$

where Nf_i is the failures found on operational profile *i*, and N_i is the total number of test cases for operational profile *i*.

We can compare the reliability of different operational profiles by applying the above formula. Similarly, the other reliabilities can be estimated as follows:

$$R_0 = 1 - [\Sigma Nf_i / \Sigma N_i]$$

$$\Re_\mathrm{i} = 1 - [NF_i / N_i]$$

$$\Re_0 = 1 - [\Sigma NF_i / \Sigma N_i$$

where NF_i is the failure limit of the operational profile *i.*

Others have used the *acceptance chart* [15] to decide whether the software product can be rejected or accepted. We believe that there is a better way to use the acceptance criteria for making acceptance decisions. In our approach, if the number of faults found in an operational profile is below its acceptance threshold limit, the test of this operational profile is acceptable.

According to the OBT strategy, the user can decide to accept, provisionally accept, conditionally accept, send back for rework, or reject the software deliverable based on the results of UAT. Provisional or conditional acceptance occurs only when the first acceptance criterion is met and some non-critical defects are detected during UAT. We next describe the five possible acceptance decisions.

1. *Accept*
   When no defects are detected during UAT, the software deliverable should be accepted by the user.
2. *Provisional acceptance* – accepted without repair by concession
   When the reliabilities of all the operational profiles and the whole application are above the acceptance threshold limits, the software deliverable can be accepted without repair by concession if the user agrees.
3. *Conditional acceptance* – accepted with repair by concession
   When test results of the whole application are acceptable but the reliabilities of some operational profiles are below the acceptance threshold limits, the deliverable can be conditionally accepted with repair of those failed operational profiles subjected to the user's concession. The developer must fix identified faults corresponding to those failed operational profiles according to their fixing priorities and within an agreed period of time; otherwise, the software deliverable should be rejected. The fixing priority of non-critical faults can be classified into three levels: high, medium, and low. High priority faults should be fixed before other faults.

4. *Rework to meet specified requirements*
   When there are too many faults found for the whole application, but the reliabilities of individual operational profiles are acceptable, then rework is needed. The software deliverable can be considered for acceptance only when it is retrofitted and retested until the total number of faults fall within the threshold limit for the whole application.
5. *Rejected*
   When there are too many faults found in both the whole application and the individual operational profiles, the software deliverable should be rejected, even if all detected faults are non-critical.

Table 4 summarizes the actions taken under the five scenarios described above.

Table 4. Acceptance decisions

Case	$R_i \geq \Re_i \quad \forall i$	$R_0 \geq \Re_o$	*Fault handling resolution*
1	No fault found		*Accept*
2	Y	Y	*Provisional acceptance*
3	N	Y	*Conditional acceptance*
4	Y	N	*Rework*
5	N	N	*Rejected*

Figure 1 shows the steps in accepting a software deliverable. First, the acceptance test cases are run. Whenever a critical defect is detected, the software deliverable is rejected. After executing all test cases, the number of non-critical defects is checked against the acceptance threshold limit of each operational profile class and against the acceptance threshold limit of the whole application. Depending on the different scenarios, one of the five cases presented in Table 4 can be applied.

3.3.4 *Compliant with ISO9001 test requirements*

The OBT strategy satisfied the requirements of the ISO9001 on software testing. This section highlights how it addresses the key ISO9001 test requirements.

Regarding software acceptance according to Section 4.10 of ISO9001, the OBT strategy demonstrates conformance to the user's expectations by focusing on two major areas: user's inspection of prototypes at regular intervals, and acceptance testing with user's involvement to verify program correctness from the user's perspectives.

At the appropriate intervals during the software development cycle, the program prototype should be demonstrated to the user to verify conformance to the user's expectation. As prototyping encourages feedback from the user, the developer should produce a simple working model which can then be expanded and refined.

According to Section 4.10.4 of ISO9001, 'Supplier shall carry out all final inspection and testing (UAT) in accordance with quality plan / documented procedures to complete evidence of software deliverable conforming to the requirements specified on the documentation.' These test procedures and the documentation should be formally specified and agreed by the developer and user before the start of inspection and testing. The OBT strategy requires the following test documentation: test plan, test specification, test case database, test records and test review reports. This set of documentation satisfies the ISO 9000-3 requirements on documentation.

ISO9001 also emphasizes documented procedures. In particular, Section 4.10 of ISO9001 requires the software developer to establish and maintain documented procedures for inspection and testing activities. Our strategy includes review of the test plans, test specifications, and test procedures as the first steps of the testing activities.

The OBT strategy also satisfies the ISO9000 requirements on product measurement: 'Some metrics should be used which represent the reported field failures and/or defects from the customer's viewpoint. .... The supplier of software produce should collect and act on quantitative measures of the quality of these software products.' (ISO9000-3, Section 6.4.1). The acceptance criteria of the OBT strategy can serve as such a product measurement.

Also, as described in Section 3.3.3, the OBT strategy includes detailed acceptance guidelines for all combinations of fault handling. This is compliant with Section 4.13.2 of ISO9001 on discrepancies handling.

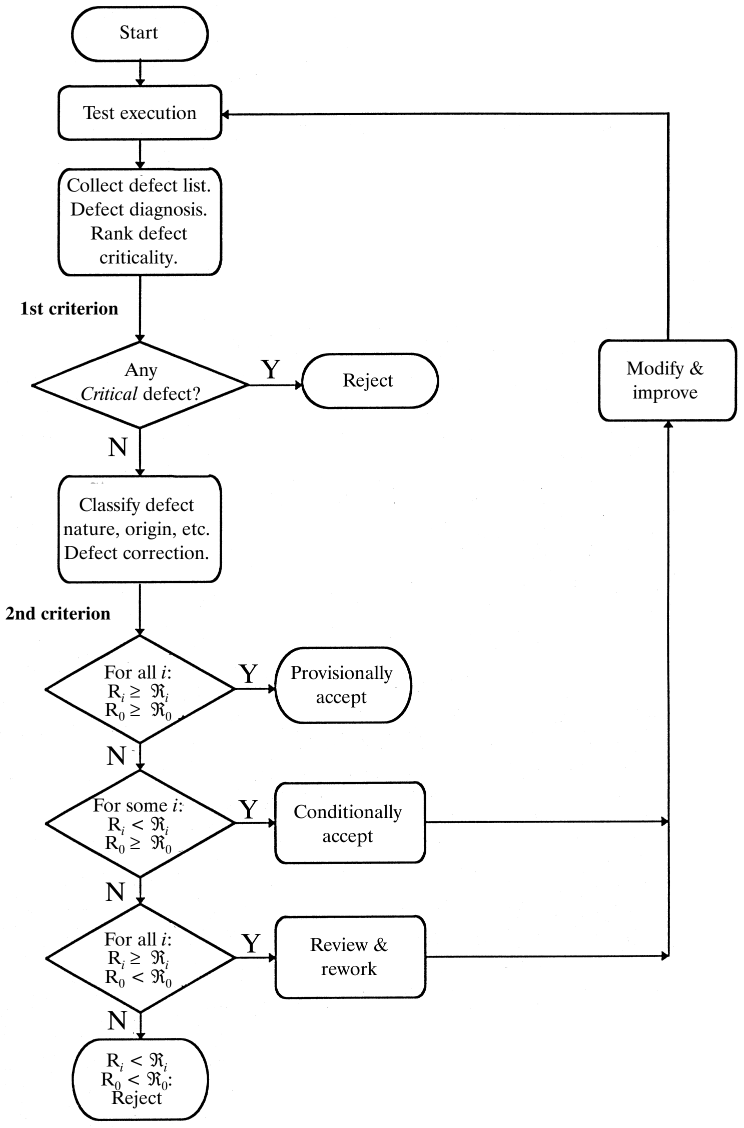
Start
Test execution
Collect defect list. Defect diagnosis. Rank defect criticality.
1st criterion
Any *Critical* defect?
Y
Reject
N
Modify & improve
Classify defect nature, origin, etc. Defect correction.
2nd criterion
For all i: $R_i \geq \Re_i$ $R_0 \geq \Re_0$
Y
Provisionally accept
N
For some i: $R_i < \Re_i$ $R_0 \geq \Re_0$
Y
Conditionally accept
N
For all i: $R_i \geq \Re_i$ $R_0 < \Re_0$
Y
Review & rework
N
$R_i < \Re_i$ $R_0 < \Re_0$: Reject

Fig.1. Applying the acceptance criteria

4. Evaluation of UAT strategies

Compared to the behaviour-based and black-box approaches for UAT, the strengths of the OBT strategy include:

1. As testing is driven by the operational profile, it is very efficient in detecting failures associated with the most frequently used operations.
2. The partnership of the developer and user (who is the application domain expert [17]) working together in designing test cases has the following advantages:
   (a) the operational profile constrains the input domain and helps to reduce the number of test cases;
   (b) more testing is done on those system operations that are more likely to be used during actual system operation.
3. As the OBT strategy satisfies the ISO9001 requirement, it provides a comprehensive UAT approach with well-defined criteria for software acceptance.
4. The OBT strategy provides a pragmatic use of the ISO9001 standard for software development. For software developers, the strategy not only enables them to interpret the requirements of ISO9001 standard during certification preparation but also can be used for software process improvement.

However, the OBT strategy requires more analysis upfront as the user needs to determine the different operational profiles. Also, test selection involves more work as the frequency of occurrence becomes a factor in picking the test cases.

Table 5 compares the three UAT strategies.

Table 5. Comparing the UAT strategies

Evaluation criteria	*Behaviour-based strategy*	*Black-box approach*	*Operation-based strategy*
Test approach	Scenario-analysis/FSM approach	Base on external functionality	Create test case for each user operational profile Testing based on probability of occurrence
Test oracle	System specification	User requirement, procedural manual	User requirement
Acceptance criteria	Not specified	No major problem found	1. No critical fault remains 2. Acceptable reliability
User involvement	Very limited	Medium	Medium
Fault handling	Not specified	Faults list	Well-defined procedure
Strengths	Consider external behaviour/scenarios	Operational procedures are incorporated into testing	1. Capture user's perspectives 2. Acceptance criteria are clearly defined 3. Complaint to ISO9001 requirements
Weaknesses	Time consuming, requires technical expertise	Lack of acceptance criteria	More analysis upfront

5. Conclusions

The user acceptance test is an important step as the last line of verification to check the readiness of a software deliverable against the user's expectation, yet there has been little research done in this area. Software reliability engineering has been successfully applied by many organizations to assist in making software release decisions. It is expected that this proven technique can also be used to add more rigour to the often *ad hoc* approach to UAT.

In this paper, we first identify the uniqueness of UAT relative to other test phases. We then propose a new strategy which includes an operational profile in the test design, and takes into consideration the software test requirements of ISO9001. Our strategy also clearly defines the acceptance criteria and a set of follow-up actions based on the test results. The user is provided with an objective guideline to assist in the acceptance of software deliverables.

We plan to apply our strategy to a real-life application to evaluate its effectiveness and strengths compared to other UAT strategies. One difficulty that we anticipate is the availability of user operational profiles. Unless the user is willing to develop the various operational profiles, the OBT strategy cannot be effectively applied.

We also intend to develop a tool to assist the user in the creation of the operational profiles. This tool will compute the different user operational profiles and determine the number of test cases for each profile based on its weighted criticality. The tool can also use the user provided acceptance threshold limits to recommend an appropriate acceptance decision.

References

1. R. Dromey. A Model for Software Product Quality, *IEEE Trans. Software Eng*., **SE-21** (2), (1995), pp. 146–162.
2. B. Hetzel. *The Complete Guide to Software Testing*, 2nd edn (Wiley-QED, 1988).
3. P. Hsia, J. Gao, J. Samuel, D. Kung, Y. Toyoshima and C. Chen. Behavior-based acceptance testing of software systems : a formal scenario approach, *Proceedings of the 18th Annual International Computer Software and Applications Conference (COMPSAC)*, 1994.
4. J. Musa, A. Iannino and K. Okumoto, K. *Software Reliability: Professional Edition* (McGraw Hill, 1990).
5. P. Schroeder and B. Korel. Improving testing efficiency using structural analysis, Department of Computer Science, Illinois Institute of Technology, 1995.
6. *Software Engineering Encyclopedia* (McGraw Hill, 1994).
7. IEEE Std. 610.12 – 1990. IEEE Standard Glossary of Software Engineering Terminology.
8. B. Beizer. *Software Testing Techniques*, 2nd edn (Van Nostrand Reinhold, 1990).
9. W.E. Howden. A survey of static analysis methods, in E. Miller and W.E. Howden (eds) *Software Testing and Validation Techniques*, 2nd edn (IEEE Computer Society Press, 1981) pp. 101–115.
10. *SSADM and Information Systems Procurement, Information Systems Engineering Library* (CTTA, 1994).
11. H. Leung and L. White. A study of integration testing and software regression at the integration level, *Proceedings of a Conference on Software Maintenance*, Nov. 1990, pp. 290–301 (IEEE Computer Society Press).
12. J. Musa. Operational profiles in software-reliability engineering. *IEEE Software,* **10**, March (1993) pp. 14–32.
13. M. Chandrasekharan, B. Dasarathy and Z. Kishimot. Requirements-based testing of real-time systems : modeling for testability, *IEEE Computer*, April (1995) pp. 71–80.
14. C. Ramamoorthy and F. Bastani. Software reliability – status and perspectives, *IEEE Trans. on Software Eng*., **SE-8** (4) (1982) pp. 354–371.
15. J. Musa and F. Ackerman. Quantifying software validation: when to stop testing, *IEEE Software*, **6**, May (1989) pp. 19–27.
16. W. Ehrlich, J. Stempfel, R. Wu. Application of software reliability modeling to product quality and test process, *Proceedings 12th International Conference on Software Engineering*, Nice, France, March 1990 (IEEE Computer Society Press) pp. 108–116.

17. J. Ning. User involvement in software re-engineering, *Proceedings of the 14th Annual International Computer Software Application Conference (COMPSAC)*, 1990, pp. 615–616.
18. J.D. Musa. Software-reliability-engineered testing. *IEEE Computer*, November (1996), pp. 61–68.