NBSIR 86-3407

An Experiment in Software Acceptance Testing

Dolores R. Wallace

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
Institute for Computer Sciences and Technology
Gaithersburg, MD 20899

July 1986

U.S. DEPARTMENT OF COMMERCE

NATIONAL BUREAU OF STANDARDS

NBSIR 86-3407

AN EXPERIMENT IN SOFTWARE ACCEPTANCE TESTING

Dolores R. Wallace

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
Institute for Computer Sciences and Technology
Gaithersburg, MD 20899

June 1986

U.S. DEPARTMENT OF COMMERCE, Malcolm Baldrige, *Secretary*
NATIONAL BUREAU OF STANDARDS, Ernest Ambler, *Director*

AN EXPERIMENT IN SOFTWARE ACCEPTANCE TESTING

Dolores R. Wallace

ABSTRACT

Software acceptance testing was performed on a prototype software engineering environment as part of the program to provide information to Federal agencies for improving quality and productivity in software development and maintenance. The purpose of software acceptance testing is to demonstrate to its purchasers that the software satisfies its requirements. This report describes the method and standards applied in this study in software acceptance testing. The report also discusses the difficulties encountered during the study and proposes research directions for software acceptance testing.

KEYWORDS

software acceptance criteria; software acceptance testing; software engineering environment; software standards; software test planning; software test tools.

FOREWORD

Under the Brooks Act, the National Bureau of Standards Institute for Computer Sciences and Technology (ICST) promotes the cost effective selection, acquisition, and utilization of automatic data processing resources within Federal agencies. ICST efforts include research in computer science and technology, direct technical assistance, and the development of Federal standards for data processing equipment, practices, and software.

ICST has published several documents on software verification, validation and testing as part of this responsibility. This report presents the results of a study in software acceptance testing performed on an experimental software engineering environment. It discusses applicable software standards, acceptance criteria, test procedures and project difficulties to enable readers to gain an understanding of the software acceptance test process.

The experimental software engineering environment prototype and other commercial products are identified in this paper for clarification of specific concepts. In no case does such identification imply recommendation or endorsement by the National Bureau of Standards, nor does it imply that the material identified is necessarily the best for the purpose.

TABLE OF CONTENTS

1.0	INTRODUCTION	1
2.0	DESCRIPTION OF THE TEST EXPERIMENT	1
3.0	SOFTWARE ENGINEERING GUIDANCE	2
4.0	TEST REQUIREMENTS & ACCEPTANCE CRITERIA	4
5.0	TEST PROCEDURE	6
6.0	PROJECT DIFFICULTIES	8
7.0	PROJECT EVALUATION	10
8.0	FUTURE RESEARCH	11
9.0	REFERENCES	12

LIST OF FIGURES

Figure 1.	Characteristics of selected guidance documents	3
Figure 2.	Project requirements for acceptance testing	5
Figure 3.	Project procedure for acceptance testing	6
Figure 4.	Difficulties of acceptance test project	8
Figure 5.	Benefits of acceptance test project	10
Figure 6.	Definition of research topics	12

1.0 INTRODUCTION

Software acceptance testing is usually performed to enable a customer to determine whether or not to accept a software system. In our experiment we performed software acceptance testing for an additional purpose: to define areas in which research and guidance are needed in software acceptance testing. We examined current software engineering guidance and traditional acceptance test procedures for their applicability to the acceptance testing of software for which we were not the original customer.

As part of its research on characteristics and functions of software engineering environments, the Institute for Computer Sciences and Technology (ICST) of the National Bureau of Standards (NBS) participated in the evaluation of a modern software development environment both to provide feedback to the developers and to gain first-hand experience with the tools and uses of environments [WAL86a]. This project also provided the opportunity to apply current software validation, verification and testing (VV&T) guidance and practices for the purpose of studying their strengths and weaknesses.

While the test experiment was underway, two other ICST documents were being prepared; each of these has influenced, and has been influenced by, the test experiment. One document describes the characteristics and functions of software engineering environments and provides examples of their implementation in environments [HOUG85]. The other document discusses traditional acceptance test procedures [WAL86b] and serves as the basis on which the results of this test experiment will build.

The software development environment that was examined is the result of an ongoing research project. The goal of the project's developers was to study some of the central questions confronting the would-be builder of a software development environment by creating and analyzing an experimental prototype of a useful environment [OSTE83]. Preliminary versions of this environment, called Toolpack, were released for test and evaluation during 1984.

The Toolpack support environment was developed for a target community of developers of mathematical software written in FORTRAN 77, to support the development and the verification, validation, and testing activities of the coding phase of the software lifecycle.

Toolpack is a collaborative effort involving researchers at Argonne National Laboratories, Bell Communications Research,Inc., Morristown, NJ, Jet Propulsion Laboratory, Numerical Algorithms Group, Ltd., Purdue University, University of Arizona at Tucson, and the University of Colorado at Boulder. Funding in the United States has been supplied by the National Science Foundation and the Department of Energy; funding in England has been supplied by Numerical Algorithms Group, Ltd.

Under no circumstances does this report constitute an endorsement of Toolpack by the National Bureau of Standards; nor does it imply that Toolpack is necessarily the best product for its purpose. It is also understood that the product assessed is still undergoing development, that it is intended to be a research product, and that many features reported herein are likely to be different in future versions of Toolpack.

2.0 DESCRIPTION OF THE TEST EXPERIMENT

The Toolpack beta-test project also provided the opportunity to develop a case study in software acceptance testing. Our goal was not to develop new techniques for acceptance testing but rather to examine and use the procedures indicated in current software engineering documents and then define areas where research and guidance in acceptance testing are needed.

This case study was a real world situation in which we encountered several classic problems. The product was not complete (from the beta-test and research perspectives that was a perfectly acceptable circumstance.) We were not the original purchasers and therefore had not specified the requirements for the product. In our role, we acted as potential buyers who needed to examine this system for acceptance, i.e., that it would satisfy our requirements. Hence, the pre-test questions:

What existing guidance in software testing can help us develop and implement an effective test plan to aid in our acceptance decision?

How do we identify our own requirements in quantitative terms that enable us to show that the software system meets our needs?

How do we define acceptance criteria for state-of-the-art characteristics?

How do we implement our test plan? That is, what procedures are necessary?

Some post-test questions are:

What problems did we encounter?

Did we achieve the test goals from the perspective of the software tools and environments program?

Did we achieve our goals for the software VV&T program?

What did we learn about software engineering guidance?

What areas in testing require further research and guidance?

3.0 SOFTWARE ENGINEERING GUIDANCE

From our examination of literature on software engineering and software VV&T, we found a few software engineering standards and guidelines that at least mentioned software acceptance testing. Since we wanted to locate weaknesses and strengths of existing standards and the areas where additional guidance is needed, we attempted to restrict ourselves to planning the project based on information from these documents, whose characteristics are given in Figure 1.

For this study, the appropriate guidance came from the following documents:

o FIPS38, Guideline for Documentation of Computer Programs and Automated Data Systems [FIPS38].

o FIPS101, Guideline for Lifecycle Validation, Verification, and Testing of Computer Software [FIPS101].

o ANSI/IEEE STD 829-1983 IEEE Standard for Software Test Documentation [IEEE829].

o IEEE P1012, Draft Standard for Software Verification and Validation Plans [FUJI85].

FIPS38 views the test plan as a comprehensive document that itemizes all information prior to executing the tests. This document is completed during the programming phase and contains all the information on test cases, specifications, and procedures.

FIPS101 addresses acceptance testing as part of the development project. Acceptance criteria are developed during the requirements specification of the project by both

Major Features

o Testing defined generically

o Planning heavily emphasized

o Documenting, not how to perform

o Tailoring to project needs

o Assuming user involvement in the software process

Figure 1. Characteristics of selected guidance documents

developer and purchaser. The test plan is written during the requirements phase. Later documents specify the actual test case designs, specifications, and procedures. The developer may be the installer of the software and may perform at least a basic acceptance test for the purchaser. Then the purchaser may perform additional tests. The acceptance criteria are clearly-defined, with user functional and performance requirements expected to be satisfied.

ANSI/IEEE Std 829-1983 describes a comprehensive test document set that may apply to any test type. It views the test plan as an initial guiding document for the test process. The test plan provides the information necessary to manage the testing but not to implement the testing. That information appears in separate documents for test design, test case, and test procedures. Test reporting is covered by four separate documents.

The IEEE P1012 draft standard assumes custom built software with the purchaser participating in specifying the requirements. This draft standard requires an acceptance test plan early in the development of the product. It assumes that a requirements document and user documentation are available to those planning the acceptance testing. It also requires a detailed description of the generic types of information (scheduling, organizing, test approach, tool needs, etc) that any test plan should contain. Like FIPS101 and IEEE 829, the IEEE P1012 draft views the test plan as a planning document, with the details for the implementation of the tests appearing in the test case designs, specifications, and procedures.

None of the above documents completely defines how to perform acceptance testing. Two (FIPS101 and IEEE P1012) specifically mention acceptance testing, and in them there is an implicit assumption that their guidance has been followed. The documents imply that how a VV&T program has been implemented during development may affect some of the acceptance test considerations. In some cases, the customer is involved early in the development of the product to establish the acceptance criteria. For all software, prior to acceptance test, the software is expected to be complete, with adequate documentation for determining test requirements.

The concept of "tailoring" is emphasized in FIPS101; that is, a VV&T program should be built from the framework of FIPS101 to satisfy a project's particular needs.

We extrapolated this approach to the project by tailoring the plan outlines and suggested contents to define our acceptance test plan.

In summary, the guidance documents provide an overview of testing with general formats for test plans, but not a minimum level of content for acceptance test plans. At the time of this experiment, none of the documents addressed the different goals and objectives of different types of testing[1] . Other topics include the need for management and planning, the types of documentation needed for complete testing, test procedures, and the need for quantitative criteria on which to evaluate test results. The guidance is applicable to custom built software in development or maintenance. Those who acquire, manage, develop, maintain, vend, and use software are involved in software acceptance testing. We hoped our experiment would be a first step leading to more complete guidance in software acceptance testing for this audience.

4.0 TEST REQUIREMENTS & ACCEPTANCE CRITERIA

Acceptance testing is performed to demonstrate that the final software product performs as the customer expects it to, based on customer requirements. As researchers in software tools, however, we weren't as concerned with how well a particular tool, or set of tools, behaved in this particular system. It was more important to determine the potential value of a tool or set of tools. We had to design the test requirements to satisfy both needs.

We were not involved in the evolution of the system; we did not know precisely the original requirements for the system. Our approach was to define Toolpack's functional requirements from the available literature and to assess how well the software met those requirements.

Designing the test requirements to answer questions of the tools research program required scenario-building, that is, determining how Toolpack would most likely be used by its intended audience. When customers have not written the original software requirements, they must define completely the capabilities expected of the system. The customers and the acceptance test planners (who may be the same) design tests that implement these capabilities as they would be used by the customers. In this case, that meant developing tests that would exercise the processes a programmer uses to develop, test, and maintain a computer program. This scenario-building technique enables customers to determine if a software product will satisfy their unique situation. The remaining test requirements described in this report resulted from this scenario-building.

From the perspective of our software tools and environments program, we selected characteristics and functions for evaluation to determine how well they would aid in software development or maintenance. As purchasers, we wanted to know how the system would respond in our operating environment. Examples of questions that were evaluated include: Would its use disrupt other users on our computer? Would its use detract from or add to users' access to the capabilities of the computer's operating system? Would more than one copy of Toolpack reside efficiently on our system?

The guidance we were following is applicable throughout the development of the software. We weren't involved in that development; hence, some of the assurances of the development's VV&T program weren't available to us. We expanded the traditional definition of acceptance testing to allow us to consider some of these quality issues. We

[1]At a later meeting (March 1985) of the P1012 group, the group added specific definitions of design, integration, system and acceptance test to the draft standard.

would want to know something about future maintenance costs, whether or not we had the source code. Therefore, we needed to perform some maintainability measurements. We asked the questions that we would expect customers to ask before they purchase a product. Time constraints prevented us from examining as many quality concerns as we would have liked.

General categories which provided the basis for the test and evaluation requirements and cases are summarized in Figure 2. Specific requirements, including acceptance criteria, and test cases were included in the project test plan.

General Test and Evaluation Categories

o evidence that Toolpack meets its requirements as defined in the technical literature

o overall performance as a complete, integrated software engineering environment

o assurance that the product has been verified and validated

o information concerning coding standards, procedures, etc. used by developers

o how well Toolpack would operate on our computer

o how Toolpack operation would affect users of other software on our computer

o maintainability of Toolpack

Figure 2. Project requirements for acceptance testing

In defining test and evaluation requirements we took advantage of our access to the source code. To evaluate for maintainability, we examined the code for features in a maintenance guideline [FIPS106] and executed measures requiring source code. Without source code, an acceptance criterion might have been evidence from the developer that maintainability measurements have been met (e.g., statistics on module size, proof of conformance to a structured language standard).

In some cases, acceptance criteria were easy to define, (e.g., either the function would execute as defined or it would fail.) In others, such as for quality or state-of-the-art features, defining criteria was more complex. A feature such as maintainability is difficult to define in quantitative terms. A judgment on the completeness of the tool set or the value of individual tools may also be considered subjective. To maintain objectivity, we decided to use accepted guidelines or other technical information. For maintainability, we examined the Toolpack code according to [FIPS106]. In order to evaluate the completeness of the environment, we checked against the levels of capability as presented in [BRAN81]. These levels were particularly suitable as a baseline because as the support level increased, so did the level of integration, another of the proposed features of Toolpack. While this project was underway, a technical journal presented the results of a different type of study on the value of software tools to the programmer [HANS85]. We compared Toolpack's tools against the conclusions of this study also.

5.0 TEST PROCEDURE

In order to perform the acceptance testing, we established a general procedure to organize the various tasks and to allow us to evaluate how large the effort would be. The procedure list, shown in Figure 3, does not cover all tasks, such as the initial decision to apply existing VV&T guidance.

In general, we were able to follow this procedure. Although we did everything on the list, some steps were performed out of sequence. These include the management review, some significant iterations to the test plan, and the document examination. For an informal review, an impartial and highly experienced staff member provided comments on the first draft of the plan. For the formal review, we had difficulty getting management staff together. When it did occur, near the end of testing, it became a review of the project. The nature of comments was such that had this review occurred earlier, the project might have been easier.

The iterative nature of test planning was evident on this project. As a consequence of the management review of the test plan, we added more information concerning the automation of the testing process and descriptions of procedures which we were actually implementing but had not documented in the plan. The original test plan was

---

PROCEDURE for EXPERIMENTAL PROJECT

o develop a test plan including
  - management information
  - the general approach to testing
  - functional requirements and general test requirements
  - criteria for acceptance

o conduct a management review of the test plan

o develop test design and cases for each requirement from the perspective of the original customer

o develop other requirements to test needs as potential customers (scenario-building)

o develop specific test procedures

o develop algorithms, software tools for measurements and data collection

o define other analysis methods

o build a test library

o execute tests

o maintain a test log, including an automated history profile

o evaluate and document.

Figure 3. Project procedure for acceptance testing

written before the results of another tool study [HANS85] were published. Although we were evaluating completeness of the environment according to integrated levels of capability suggested by [BRAN81], this article presented a different perspective for measuring the worth of individual tools. We wanted to include this approach in our work and modified the test plan once more.

The test plan called for an evaluation of the Toolpack documents. Because we needed to read all the literature in order to get started, we considered the initial reading to be the required examination. Later, near the end of the project, we briefly re-visited the documents. On a larger project, a more formal examination may have been required.

After installation and preliminary examination, we knew which Toolpack functions existed in our configuration. We used those capabilities to examine their usefulness in a programming environment and to determine how well they met the level of expectation. In addition, we evaluated properties of the future complete system as it was described. Obviously, we could not try these properties ourselves; rather we evaluated against our own and others' experiences. In one case, we had already examined a feature not included in our configuration, that is, a command interpreter [CLEM84] which was a prototype of Toolpack's command interpreter, the tool that would complete the integration of the Toolpack tools. We considered this to be at least a partial "proof of concept" of tool integration.

For some of the test requirements, we had to develop software. We wanted to keep our development efforts as simple as possible, using readily available technology. Software was needed to calculate storage and timing requirements, to test the extensibility feature, that is, the capability to insert additional tools into the system, and to partially automate the testing process. We developed a test library of programs to test the capabilities of Toolpack and the results of the various tests. The names of test programs and test results in the test library provided a cross-reference between them.

Programs in the test library were prepared to measure execution time on programs of various lengths, to test functional capabilities such as syntax error detection, data-flow analysis, structuring, and compatibility with FORTRAN 66. (The last requirement was included because potential users of Toolpack may have many programs in FORTRAN 66.) A sample program was written to demonstrate Toolpack's extensibility feature, the ease with which a tool can be added.

Toolpack provides access to its tools in different modes. In embedded mode the Toolpack user operates completely within the Toolpack environment and can access features outside Toolpack only by completely exiting the environment. In the stand-alone mode, the user has access to individual Toolpack tools but control is maintained by the computer's operating system. To test and record an embedded mode session, the command executor was run with results recorded by the "script" function of UNIX[2] . Stand-alone test scripts were designed to automatically execute tools on a collection of test files. These test sessions were also recorded using UNIX "script". The test files were provided one-by-one to the program being tested. After conducting the tests we were able to incorporate portions of the log files into test reports.

Output of the UNIX function "script" provides history files of the test execution. However, test summary reports are also needed. To capture our impressions

[2]UNIX is a trademark of AT&T.

immediately, we wanted to create a "notes" file and edit it inside Toolpack. Technical difficulties forced us to make manual notes which we later put together as one automated file.

6.0 PROJECT DIFFICULTIES

Some project difficulties were easy to handle; others would make reasonable research projects. We were not immune to the problems of real life. Some of these difficulties are listed in Figure 4.

- o personnel turnover
- o various configurations of the product to test
- o separation of developmental from acceptance tests
- o determining requirements for purchased rather than custom built software
- o defining acceptance criteria for state of art features
- o loss of objectivity in testing and evaluating
- o realization of most of the anticipated constraints
- o lack of automated data collection facilities
- o too many iterations to add more test cases
- o inadequate configuration management for the test items.

Figure 4. Difficulties of acceptance test project

During this project, staff changes meant reorganization, delays, and changes in focus. The scope of the project was expanded to satify goals of both the VV&T and softare engineering environments projects.

We had begun our preparations for evaluating Toolpack long before the first system configuration arrived. We received several configurations of the Toolpack system, but never received the complete system. The test plan, based on descriptions of Toolpack from the technical literature and some Toolpack documents published by the researchers, was based on the complete system and could not be implemented completely. The latest version we received had more tools, still in the experimental stage, and perhaps represented the intended prototype more closely than previous versions. The expectation (of developers and testers) that the new tools would not perform well meant that this version was less likely to perform overall as well as the version preceding it. For the formal testing, we selected the earlier, less complete environment because its tools were more likely to perform well and to indicate more clearly how they would benefit a programmer. This selection probably reduced our objectivity.

Acceptance testing is usually performed on the final product, often in the operating environment. In many cases the customers do not further examine the source code itself. However, we did have the source code and it was very difficult to separate acceptance testing from the types of testing and measurements that might be performed during development. Some measurements on the code would answer questions concerning how the tools were created and how they were integrated. In the original specifications, there may have been requirements for performance and maintainability whose fulfillment might have been verified during development for the original customer. For customers not involved in the development of a product, some acceptance criteria might require evidence that the source code was developed in accordance with standards on programming languages, VV&T, and maintenance.

Closely related to the distinctions between developmental and acceptance testing are also the distinctions between the test concerns of the original customer and those of a purchaser of completed software. In the former case, the test requirements are built around the requirements for the software system, which the original customer specified. In the second case, the test requirements are designed to show that the system does not only what the developer says it does, but also that it will fulfill scenarios desired by the new customer. Scenario-building requires knowledge of the system to be tested and of how the new customer will attempt to use the system.

As acceptance testers, we tried to use quantitative criteria in order to make highly objective evaluations. But, some of the features we were looking at were new concepts; quantifying criteria for state of art characteristics involved examining other research results. From the perspective of the software tools project, we were frequently more interested in the proof of the concept of a feature. We tried to envision the tools in a completed environment and how as an integrated collection, they might help or hinder a programmer. We had to consider whether or not such an environment would in some cases be less valuable to a programmer, for example, if the programmer's operating system also included many tools. Our experience and the reference documents [BRAN81, HANS85] provided some guidance, but as proponents of automated software engineering aids, we nevertheless may have lost some objectivity during testing. In our work with the system and in our discussions about features we might like to have, we generally ended up saying something like, "Well, that can be put in or modified."

Another example is that we had previously examined ODIN [CLEM84], a command interpreter similar to the one proposed for Toolpack, the feature which would truly tie the system together. The configuration which we chose for the testing did not have the command interpreter (and the final version had only an experimental command interpreter). In ODIN we saw that the concept could work; from our research perspective that was satisfactory and in some sense met acceptance criteria. A more objective analysis might be made if the acceptance criteria left no room for such opinion, (e.g., we could look at the command interpreter only in the environment.) In our case, we brought some of this on ourselves by designing the project to have the somewhat conflicting goals of exploring research concepts and accepting them at the same time. In reality, we were testing to accept the concepts, not the environment! In any case, the testing staff should have no interest in the product other than to tell the customers whether or not it meets the acceptance criteria.

As in many real life situations, we realized several constraints. The full system never arrived; in fact we received the system in bits and pieces. Without knowing when we would have the total system, it was difficult to assign staff full-time to the evaluation project. We developed the test plan primarily from research papers rather than requirements specifications and user manuals. While installing the system and figuring out what we actually had, we essentially ran all the functional tests on an informal basis and knew the system's capabilities before the test plan was completed. Repeating all of them formally, when there are time limitations, was not reasonable, at least for achieving the goals of the tools project. As the algorithms were developed to aid in measurements, they were individually and methodically executed and recorded. Later we generated the test scripts and re-executed several tests formally. However, we had gathered most of our results informally. Because other tasks were demanding our attention, we had to use the remaining project time for documentation.

It may have been worthwhile to rerun some tests formally if we had better data collection methods. Examples are the automatic recording of screen images or the quick and efficient recording of comments without exiting the Toolpack system. We found the line editor slow and cumbersome. Files created inside Toolpack were not easily accessible outside Toolpack. It was time-consuming to exit Toolpack, write notes and then return to Toolpack. A simple exit to UNIX that didn't require closure of the system would have been a great aid. Instead we usually recorded observations on paper, then did most of our computerized documentation near the end of the project.

Initially, we didn't impose firm cut-off dates for changes to the test plan and requirements or to the stop conditions for the testing. We had several iterations to keep track of. We took advantage of the tools in our computer's operating environment to manage the various configurations. Yet, more automation was needed in the configuration management of the testing process, from plan versions through execution and evaluation of each test. In fact, the ideal situation would have been to use a complete software engineering environment for all document production, automation, and controls.

7.0 PROJECT EVALUATION

Benefits of this experiment in acceptance testing are summarized in Figure 5. Overall, we were satisfied. We collected sufficient data to evaluate the prototype Toolpack environment. In addition, we made appreciable gains in our knowledge of software engineering environments and in the types of problems encountered in developing such systems. As a result of this experiment, we are able to establish areas for more research in our software VV&T program.

- o Evaluation of the software
- o Gains in experience with software engineering environments
- o Understanding of problems in developing software engineering environments
- o Definition of problems in acceptance testing
- o Influence on a developing standard

Figure 5. Benefits of acceptance test project

An unexpected benefit of this project was the effect on an evolving software standard. As part of this project, we tried to apply appropriate sections of the draft of the IEEE Standard for Software Verification and Validation Plans (P1012), in strictly an experimental capacity. During this period we presented feedback to the P1012 working group, who then made some appropriate changes.

We learned firsthand about many of the pitfalls of software testing. We found that the guidance documents on software testing provide good general support, particularly for management and planning, but guidance is needed on how to do specific types of testing. The project also made clear that acceptance testing for non-custom built software requires attention to concerns that might have been covered during development for the original customer. The second customer's requirements may not be identical to those of the initial customer. The definition of acceptance criteria may require scenario-building in the new customer's environment, and may require more compromising of the requirements before accept or reject criteria are established.

Guidance documents appear to be written for the ideal, rather than the real life, circumstances software projects are likely to encounter. There is a need to have guidance that aids testers to reach their test goals in spite of realized constraints. The guidance should not be rigid but should provide a framework from which the users can tailor to fit their project. We expected to make changes in our plans but the frequency and number of changes indicated that we needed either to allow more lead time before testing or to state clearly in our plan that there would be changes. Guidance for planning documents should emphasize the iterative effort required between defining test requirements and implementing those requirements.

It would be useful to have examples included in guidance, but practical examples can be produced only from careful data collection during testing projects. Data collection was one of our problems. We wanted to document our results as we accrued them but two separate problems here made documentation difficult: the informal testing that preceded, and in some cases replaced, formal testing; and lack of tools to collect the test results and immediately analyze and document them.

Formal procedures for automation should be established as a general rule in one's working environment. Then, even if one is running some informal tests, automated recording procedures are in place. In essence, "informal" testing would not occur.

In some cases we were able to take advantage of the UNIX function "script" that captures all screen activity. However, for additional annotation we needed to leave Toolpack to use a screen editor. In spite of our good intentions, we usually did our writing after a complete series of tests. If we were to do further work in test automation, a good place to contribute is in providing the capability to comment interactively on tests, particularly those that involve screen responses. Our immediate reactions would have been easily saved for review for our final evaluation.

Finally, objectivity is sometimes hard to maintain. A very important element in testing is to define the acceptance criteria in quantitative terms. Those that may require subjective judgment should at least have the judgment supported by some accepted standard of performance, whether it is a formal standard or a technical evaluation by a recognized authority. These types of considerations will help to determine if the particular aspect of the software has passed or failed the acceptance criteria.

This study pointed out weaknesses in existing standards and guidelines in software acceptance testing. This study provided the information necessary to map out a project leading to a stronger set of guidelines for software acceptance testing.

8.0 FUTURE RESEARCH

One of the goals of this project was to define areas of needed research and guidance for software VV&T. Some topics that might be considered for future research are summarized in Figure 6. Future guidance should also provide management direction in determining the scope and appropriateness of the acceptance testing project for a software system.

Research in automated support should particularly emphasize methods of data collection during the testing. During testing easy access to information about Toolpack that we had already collected would have been useful; instead we hd to leave Toolpack to gain access to information stored in file system. We were able to implement some test harnessing, that is, the automated organization of the test cases and their execution, because of the powerful operating system we were using. More configuration control

Suggested Research Topics

- o Automated support
- o Software purchase vs custom build
- o Acceptance criteria
- o Test case selection.

Figure 6. Definition of research topics

mechanisms would have been useful. These experiences indicate that the entire test process can be vastly improved with automation from test planning through execution, including mechanisms to trace software requirements through test execution.

The purchasers of software systems are recognizing that much of the software they need already exists, or is in the process of being built for another customer. If they wish to purchase this software, they need methods of determining how well the software will meet their own needs. Acceptance tests for this software may include areas outside the domain of traditional software acceptance testing. Guidance to define the concerns for this type of software testing would be very useful.

Although acceptance criteria are sometimes difficult to define in quantitative terms, they are essential in helping purchasers make unbiased accept or reject decisions. Guidance is sorely needed to emphasize this. Methods to establish criteria, particularly when the software is either state-of-the-art or performing a function that is not well-known, need to be studied. Guidance is also needed to map criteria against various software products and the methods of demonstrating that each product meets its criteria.

The problem of test case selection is important for all types of testing. Basically, guidance is needed in two major areas, one technical and one managerial. The technical question for acceptance testing is how to build and select scenarios that reasonably represent the purchaser's intended use of the system. Of course, the purchaser needs to represent this intended use without doing exhaustive testing. This leads to the larger management question of how much quality assurance is enough.

This study was the first of a series of steps planned to lead to guidance in software acceptance testing. Its primary purpose was to locate the topics where more information and research are necessary before guidance can be written. The next steps are to perform the research and to locate more information concerning these topics. One approach has been to conduct a workshop to address these research questions in software acceptance testing. Future steps leading to guidance will be based on the results from this workshop.

9.0 REFERENCES

[BRAN81]

Branstad, Martha A., W. Richards Adrion, and John C. Cherniavsky, "A View of Software Development Support Systems," *Proceedings of the National Electronics Conference,* National Engineering Consortium, Inc. 1981.

[CLEM84]
Clemm, Geoffrey M., "ODIN - An Extensible Software Environment Report and User's Manual," University of Colorado, Boulder, CO, CU-CS-262-84, March, 1984.

[FIPS38]
"Guidelines for Documentation of Computer Programs and Automated Data Systems," Federal Information Processing Standards Publication 38, National Bureau of Standards, 1976.

[FIPS101]
"Guideline for Lifecycle Validation, Verification and Testing of Computer Software," Federal Information Processing Standards Publication 101, National Bureau of Standards, 1983.

[FIPS106]
"Guideline on Software Maintenance," Federal Information Processing Standards Publication 106, National Bureau of Standards, 1983.

[FUJI85]
Fujii, Roger U., Dolores R. Wallace, and Michael Edwards, "Status: Draft of the Proposed IEEE Standard for Software Verification and Validation Plans (P1012)," *Phoenix Conference on Computers and Communications,* IEEE Computer Society, NY, March, 1985.

[HANS85]
Hanson, Stephen Jose and Richard R. Rosinski, "Programmer Perceptions of Productivity and Programming Tools," *Communication of the ACM,* Vol.28, No.2, February, 1985.

[HOUG85]
Houghton, Raymond C.,Jr. and Dolores R. Wallace, "Characteristics and Functions of Software Engineering Environments," National Bureau of Standards, NBSIR 85-3250.

[IEEE829]
"IEEE Standard for Software Test Documentation," ANSI/IEEE Std.829-1983, The Institute for Electrical and Electronics Engineers, Inc., 345 East 47th SDt., New York, NY 10017.

[OSTE83]
Osterweil, Leon J., "Toolpack - An Experimental Software Development Environment Research Project," *IEEE Transactions on Software Engineering,* Vol. SE-9, No.6, November, 1983.

[WAL86a]
Wallace, Dolores R. and D. Richard Kuhn, "Study of a Prototype Software Engineering Environment," National Bureau of Standards, NBSIR 86-3408.

[WAL86b]
Wallace, Dolores R., "An Overview of Computer Software Acceptance Testing," National Bureau of Standards, NBS SP 500-136, February, 1986.

NBS-114A (REV. 2-80)

U.S. DEPT. OF COMM. **BIBLIOGRAPHIC DATA SHEET** *(See instructions)*	1. PUBLICATION OR REPORT NO. NBSIR 86-3407	2. Performing Organ. Report No.	3. Publication Date JULY 1986

4. TITLE AND SUBTITLE

An Experiment in Software Acceptance Testing

5. AUTHOR(S)

Dolores R. Wallace

6. PERFORMING ORGANIZATION *(If joint or other than NBS, see instructions)*

NATIONAL BUREAU OF STANDARDS
DEPARTMENT OF COMMERCE
WASHINGTON, D.C. 20234

7. Contract/Grant No.

8. Type of Report & Period Covered

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS *(Street, City, State, ZIP)*

10. SUPPLEMENTARY NOTES

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT *(A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)*

Software acceptance testing was performed on a prototype software engineering environment as part of the program to provide information to Federal agencies for improving quality and productivity in software development and maintenance. The purpose of software acceptance testing is to demonstrate to its purchasers that the software satisfies its requirements. This report cribes the method and standards applied in this study in software acceptance testing. The report also discusses the difficulties encountered during the study and proposes research directions for software acceptance testing.

12. KEY WORDS *(Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)*

software acceptance criteria; software acceptance testing; software engineering environment; software standards; software test planning; software test tools.

13. AVAILABILITY

[✓ X] Unlimited
☐ For Official Distribution. Do Not Release to NTIS
☐ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.
[✓ X] Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES

19

15. Price

$9.95