Techniques for Testing On-Line Interactive Programs

Kim D. Leeper
Wick Hill Associates Ltd.
Kirkland, Washington

ABSTRACT

This paper will describe various strategies for testing on-line interactive programs. These strategies include acceptance/functional testing, regression testing and contention testing. The paper will also discuss the mechanics of testing including testing by human intervention and various forms of automated testing. This information will allow you to create a viable test plan for software quality assurance in your shop.

INTRODUCTION

Program Testing. Those two words undoubtedly conjure up thoughts of long boring hours sittig in front of a terminal typing in all kinds of data looking at error messages produced by the program. This paper will present alternatives to this type of program testing. It will also describe a prototype test plan or quality assurance cycle which may provide the reader with ideas for implementing his/her own test plan for his/her own shop.

We must make sure we are all talking the same language so some definitions are in order at this point.

What is Testing?

Software testing may be thought of as a series of data items which when presented to the program under test (PUT) cause the software in question to react in a prescribed or expected fashion within its intended environment. The purpose of testing is to expose the existence of mistakes in the program or to show the absence of any such bugs. If the software does not act in the expected way then one has found a bug or mistake in the program.

Vocabulary

SCRIPT — a list of inputs or data items given to the PUT for testing purposes.

DATA CONTEX OF BUG — the collection of inputs required to cause the PUT to fail or return results which are not expected.

TYPES OF TESTING

Acceptance/Functional Testing

This type of testing is used to demonstrate that the various functions of a given software package actually works as described in its documentation. This is not exhaustive testing as it only examines one or two transactions per function. This is the typical type of testing the vast majority of users perform now.

Regression Testing

This type of testing can be used to test all the various logical paths within a given software system. Regression testing tries all the data extremes per function that the program could be expected to respond to. This type of testing is rarely performed because it is resource, that is to say hardware and personnel, intensive.

Contention Testing

This type of testing is used to determine if the database or file locking strategies that are used in your application programs actually work. Two programs are executed at the same time, one performs a transaction which locks a given item in the database. The second program attempts to access this same data that is secured by the lock via another transaction type different than the one used in the first terminal. The designer in this instance is interested in the message of action of the software to this challenge. This type of testing becomes particularly relevant when the installation has many programmers implementing many systems dealing with the same database.

THE TEST PLAN OR QUALITY ASSURANCE CYCLE

The keystone of any successful testing program is to have a viable test plan. This plan should describe all the phases a software development project goes through and then ties all the phases together in one comprehensive flow of data and actions. The plan should extensively use feedback loops so that when problems are discovered there are clear paths for the problem rectification process to follow. One possible quality assurance cycle that can be proposed may be seen in Figure 1.

The diagram indicates that the test script should be generated along with the design of the software. Many times in the design process the designer realizes some weakness in the design and will want to specify a special test in the scriptfile. S/he is encouraged to do so. Many companies that use this methodology specify programs by a test script and V/3000 screens.

Examining this diagram more closely one can see that the flow of debugging actions is closely tied to the design/maintenance of the original test script. The reason for this is to force the implementors to keep track of the bugs they discover and place them in the test script. This script should then be run against the application program whenever a new fix or correction has been applied to the original program. This script will constantly force the program to re-execute all the previous transactions which caused bugs to occur in the past, to assure the program maintenance team that no additional mistakes have been introduced by fixing the last bug.

In this version of the QA cycle the users are always in a mode of testing the delivered software. Eventually the users will find a bug which will start the whole cyclic process over again. If they don't find a bug, don't think it is not for trying. The users have eight hours per day per person to find bugs. It does not take very long before they have more execution time on the application software than the designer/implementator has. This is the time when more bugs can and will be found which will start the cycle once again.

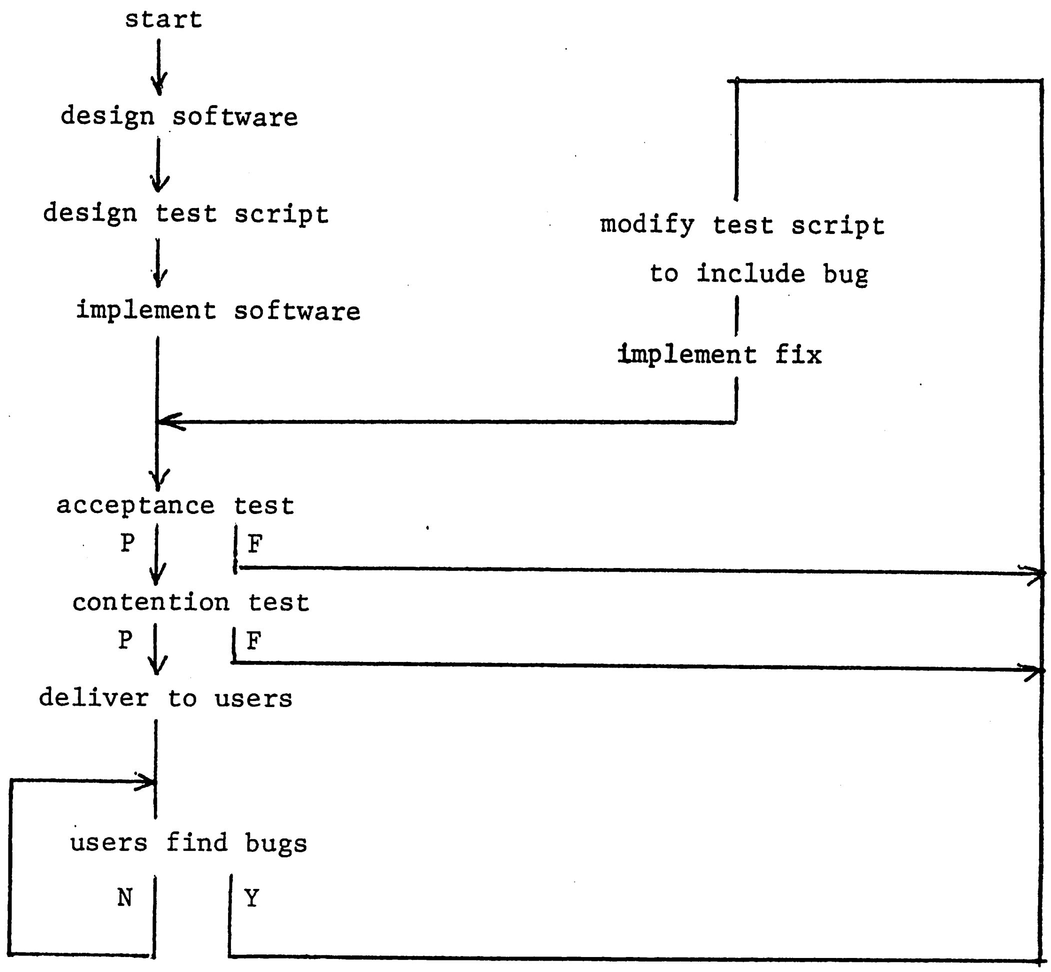


Figure 1
Quality Assurance Software Cycle

THE MECHANICS OF TESTING

Obviously, the type of testing that is currently being used is human intervention testing. This is where a programmer of analyst sits in front of a terminal and simulates a user by following a handwritten script. This approach to testing is less than desirable for a number of reasons, among those being:

1. input data error due to arrogance/boredom in applications tester;
2. non-repeatability of exact timing due to human tester;
3. the tester might not record everything happening off the screen;
4. an expensive employee is being utilized for testing purposes when s/he could be designing/ implementing more applications

A possible solution to the dilemma outlined above is to mechanially examine the software by exhaustively testing all the paths in the program by computer. Using completely random data types as input you could automate the testing process. However, as there is only so much time available during a 24 hour day it might take all day to exhaustively test a very small application program. This technique is machine bound in terms of both creating the random data and testing all the paths in the application code.

A saner approach would be to combine the above two techniques into a testing procedure that utilizes a human being's capacity for creative thought and a machine's capacity for highly efficient repetition. This technique would rest in the programmers designing the scripts used for automated testing at the same time as they design the application itself. Once the test script is produced then the machine itself tests out the application program under the watchful eye of a human. In fact the script can be used as a specification for implementing the system. As Yourdon has written, "What we are interested in is the minimum volume of test data that will adequately exercise our program."[1]

It is now possible, using VTEST/3000, to automate this testing procedure and achieve a real manner of quality control. VTEST/3000 includes full V/3000 testing capability. The compiled code runs as though it were in a live situation with VTEST/3000 providing full documentation of all errors occurring on the screen of the terminal.

In order to use VTEST effectively one must appreciate the diagram in Figure 2. There are two types of tests that VTEST can perform, block mode testing for those programs that use V/3000 and non-block mode testing for those not using V.

The first type of testing that will be discussed is non-block mode application testing. In this case VTEST looks like a non-block mode glass TTY terminal. The script file contains the actual commands and data that a user would normally type into the screen of a real terminal, everything between and including HELLO and BYE. This script file is built and maintained by the standard HP EDITOR. The script file is input to VTEST. VTEST transmits this file a line at a time to the application and VTEST prints out a report of the terminal screen before the return key was depressed and after along with the number of seconds that the response took to come back to VTEST.

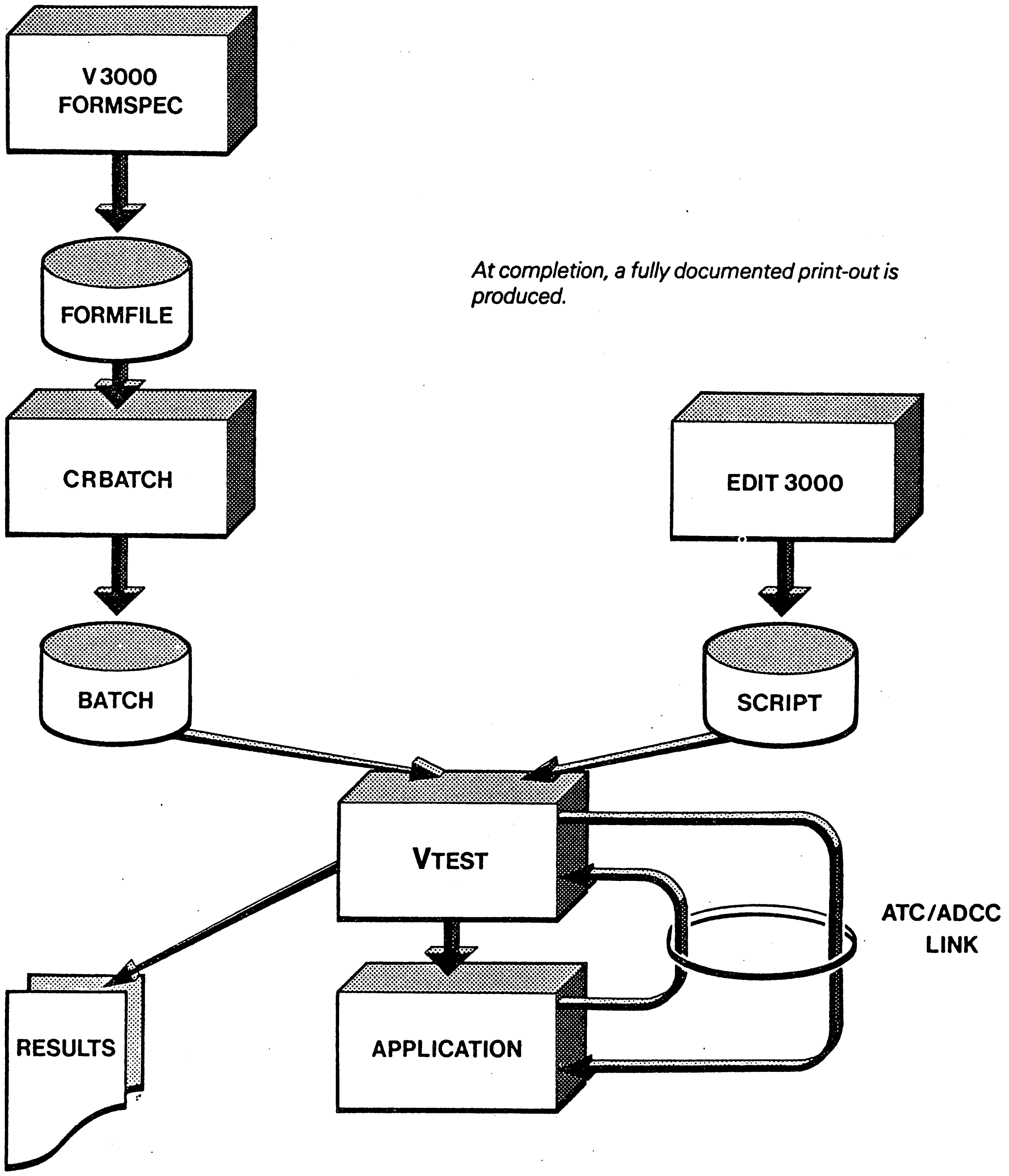
The second type of testing that will be discussed is block mode application testing. In this case VTEST looks like a HP2645 block mode terminal. The script file is the same as above with an important extension. The script file now can tell VTEST when it must transmit data to a V screen. The data for a V screen must come from a different type of file. This file is called the BATCH file. This BATCH file is created and maintained by another program called CRBATCH. CRBATCH allows the user to specify the formfile name and the form to be displayed. Data is then entered and CRBATCH reads the screen and puts the data into a BATCH file. CRBATCH allows the user to insert screens, to delete screens and modify the data in screens already in the BATCH file. It is a general purpose maintenance program or editor for BATCH files. Whenever the application program under test wants some block mode data the next record is read from the BATCH file. VTEST then transmits this record complete with all the special characters that V requires to the application. VTEST prints out a report for every transaction before the ENTER key was depressed and after the next screen was received along with the number of seconds that the response took to come back to VTEST.

One can see quite easily that VTEST fits right into a well designed quality assurance cycle.

REFERENCES

[1]Edward Yourdon, "Techniques of Program Structure and Design," Prentice-Hall, 1975.

[2]Software Research Associates, "Testing Techniques Newsletter," (415) 957-1441.



At completion, a fully documented print-out is produced.

Figure 2