

Automation in Software Testing for Military Information Systems

Jack Chandler
SSC San Diego

INTRODUCTION

This paper shows how automation can improve test results. At the beginning of this effort, a search was conducted to survey the status of automated testing. The survey revealed white papers and some commercial products that help automate testing (see Dustin [1] and Pettichord [2]). Many of the commercial products are key and cursor recorders that capture the keystrokes and cursor movements produced by test engineers during the testing process. This testing works well for testing revisions of the same product. It is not as appropriate for testing multiple pieces of software for compliance to a standard.

Dustin's paper on introducing automation to a test team states that the first phase of designing testing automation is analyzing the testing process [1]. To be of value, software testing must be a repeatable process that is definable, measurable, consistent, and objective. If the process is deficient in any of these areas, the testing will not be repeatable. This paper examines various factors in the testing process (including the human factor), describes the results of a case study on military information systems, reviews the steps required for successful automation, and provides a conclusion.

COMPONENTS OF THE SOFTWARE TESTING PROCESS

Software under Test

The most essential and basic component of the testing process is the software under test. This component cannot be changed to any great extent. The two basic categories of software under test, depending on the type of testing, are as follows: (1) testing a software product to determine whether the product is ready for release or to validate error corrections, and (2) testing multiple components of software for compliance to a standard. Both types of testing are valid; they have different requirements and different automation strategies.

Hardware for Software Testing

The second component of the testing process is the hardware platform on which the software is loaded. Improvements may be possible in this area. For example, a different hardware platform may increase the speed of software testing. Increasing the number of "seats" in use simultaneously

ABSTRACT

Software testing must be definable, measurable, consistent, and objective to be a repeatable process. This paper examines the components of the testing process, including software, hardware, the human element, and the data-collection process. It also includes a case study in test automation derived from the Defense Information Infrastructure Common Operating Environment (DII COE). A reduction in the number of human-controlled steps in the software process significantly improved test results during this case study. Automation was successful because the many different components of software were tested for compliance to a well-defined standard. Automation was straightforward because the test methodology did not require any specific assumptions about the software tested.

during the testing scenario or increasing the performance of the individual "seats" is another way hardware can improve testing. The disadvantage of substituting different hardware during the testing process is the risk of testing on a non-representative platform, which would make test results questionable. To overcome this potential problem, some testing must be done on a typical user platform.

Human Testers

The third component of the testing process is the engineer or group of engineers testing the software. Test engineers make a substantial contribution to the testing process, but the possibility of human error makes them the weakest factor in the testing process. Even more important is the fact that the process is not consistent because test engineers do not consistently make the same mistake. The most dedicated and competent engineer can err under some circumstances. Thus, eliminating the "human in the loop" can significantly improve the testing process. Reducing the number of human-controlled steps can dramatically improve software testing. An example of how to reduce the human interface areas in a testing process is described below.

Data Collection

Another major component of the software testing process is the data-collection function, which often can be improved. The data-collection function often consists of the test engineer manually filling out a paper data-collection form. The test engineer will have a test notebook or a data form in which the test data are recorded. Often, the test data are re-entered into a spreadsheet or a word processor for report generation or into an e-mail message for distribution of the test results. Forms, which provide ample opportunity for errors, could be significantly improved. For example, if the form is structured in a multiple-alternative, forced-choice paradigm rather than a less-structured essay format, subjectivity can be reduced.

Another common test procedure consists of the test engineer manually filling out an electronic data-collection form. The electronic form is better than the manual form because data are manipulated only once, thus reducing transcription errors if the data are input into a report generator or an e-mail system. Using an electronic form can require investing in more hardware to support the collection. More time may be needed to fill out the forms initially, but this method saves time by reducing or eliminating the need to transpose data.

As with paper forms, the design of the electronic form is critical. One way to improve the design is to minimize the number of probable answers while still allowing all possible answers. This is done by prompting the user to consider certain likely choices while grouping other possible answers under "other" with a space to insert a comment. A periodic review of the use of the "other" category is recommended, with the objective of providing common "other" answers with specific choices of their own.

Another useful mechanism is to collect data automatically and manually by enabling software to perform the test. Efficient results are achieved by automating to the fullest practical extent the test data acquisition process. The parts of the process that do not lend themselves to automation still

can be performed manually. A useful, proven procedure is to provide in the testing software a mechanism to input the manually derived test information. The "form" that is provided for collecting this manual information should be designed using the criteria discussed previously.

The most advanced and desirable phase of automation consists of collecting the testing data with a computer program that involves little or no human involvement. Only when the testing process is completely automated is a repeatable process achieved. Whenever a person manually performs a test, there is a chance that the test cannot be consistently repeated. Human beings are predictable in a group, but unpredictable individually.

Other Components

The other two major components of software testing are the education of the test engineers and the testing process itself.

TEST AUTOMATION: A CASE STUDY

Background

This section describes an example of end-to-end testing where the testing itself has been mostly automated and the areas that cannot be automated have been analyzed to reduce or eliminate subjectivity. The Department of Defense (DoD) has created the Defense Information Infrastructure Common Operating Environment (DII COE). Many DoD systems are being built using this "plug and play" infrastructure. The components of software for this system are called segments. A compliance specification has been created to enhance the "plug and play" capability of this infrastructure. This specification consists of over 300 requirements. A segment must pass at least the first 200+ requirements to be considered for inclusion in the DII COE.

DII COE compliance involves a time-consuming and human-intensive testing process. In one instance, about 18 person-hours were required to test a single, simple segment. Significantly greater test durations have been common in other cases. Compliance testing was a likely candidate for automation because it is common to all segments. The procedure that was used to automate this testing process is described below.

Document the Testing Process

In this step, the engineer will discover the current method of testing, including the acceptable test methods and the methods that are unsatisfactory. In many cases, the test engineer will be able to learn the test processes and procedure and to expose many inconsistencies in this step. The information that needs to be recorded during the test procedure is documented. From this experience, the test team learned that there were many "homegrown" solutions to the automation, a situation that had advantages and disadvantages. On the positive side, some work already had been completed. Unfortunately, these solutions were not consistent. After gleaning the currently automated processes, the test team captured the steps involved in those areas that were not automated.

The test team learned that not all DII COE requirements were tested. This was not because the untested requirements did not provide any added value, but rather to reduce the time required for testing. This prompted the test team to create a "best practices" spreadsheet in which

to capture the test algorithms, not the test "programs." To provide better DII COE compliance, the test team also created algorithms for the requirements that were not being tested.

Identify Common Processes

The test team identified the common processes from the algorithm spreadsheet. These will be used later to ensure that a single testing method will be used. Too often, common testing processes are coded multiple times because engineers are unaware that some of these processes are in common use. This causes inconsistencies in the use, application, and maintenance of these processes, especially if the processes need debugging or upgrade.

Design the Automation (Software)

The first steps of the automation design consisted of gathering and defining requirements. Then, the software had to be designed to meet those requirements. The design called for an object language with a compliance engine and an individual object for each requirement. Since the DII COE software is supported on multiple platforms, the test team elected to use Java as the programming language. In theory, this was expected to decrease the inconsistencies by providing a common baseline. The test team wanted to keep the design generic to accommodate any required testing process where the requirements were structured in hierarchical levels. To pass at level 5, for example, would require that all tests in levels 1 through 4 be passed as well as all tests in level 5. With this in mind, the test team designed a compliance engine with a test manager, an applicability filter, and some common data-collection agents. We defined an interface to the compliance engine that the test objects will use. The design included a report generator and a graphical user interface (GUI) that allows the test engineer to view the data-collection form and to access the various options. These components are described below in more detail.

Because many of the DII COE requirements apply only to certain types of segments, the test team needed an applicability filter to determine the applicable tests based on the segment type. Each test object specifies to the applicability filter the segment types to which it applies. The default was specified as applicable to all segment types.

The test manager launches the test objects at the appropriate times. For example, certain tests must be run while the segment is installed, whereas others cannot run until the segment is deinstalled. The test manager also runs some data-collection agents, which also must be run at certain times. The relative timing of each test is important to specify clearly when documenting the testing process.

The data-collection agents determine information about the segment under test and that segment's effect on the underlying system. The test-unique data-collection agents (if any) are common processes that collect other data from the underlying system. This feature was included in the design, not for the specific purposes of the test team, but only for the generic case.

The questionnaire object(s) obtain additional information from the test engineer. Questionnaires are presented twice in the testing process. (Figure 1 shows the testing process.) The first time the questionnaires are presented

is before the segment is installed; software developers provide this information. The second time is after installation to obtain information that the test engineer can determine easily, but that the automated software would not be able to determine (or be able to determine uniquely). The questionnaires present multiple-choice questions, including an "other" choice, where applicable. These objects read from formatted text data files and are therefore dynamic and easily modified.

The data-collection form and the menus presented are also dynamic. They are created by the compliance engine at runtime. The report generator merely specifies the state of each test object. This facilitates the additional tests as well as additional report formats. The testing process also tracks the test engineer's name and reports separately any tests that were waived or overridden. The test team found that the software to automate first should be that which provides the most efficient and largest payoff [2]. A phased approach to automation has proven most successful.

Designing the Automation Process

The requirements gathering described above also will yield one or more processes. In the beginning of software testing, all processes may not yet be in place. It is equally important to design the process. The automation software works within a process. This process will depend on the automation, which, in turn, will depend on the process recursively; often, both should be designed at the same time.

If testing has not been automated previously, the process will need a major rework. When automated testing is in place, personnel may be available to be tasked elsewhere. This will not be true in the beginning since the automation will also be undergoing testing. It is important to account for the testing assets that will be displaced by the automation. After some time, however, resource management will have to account for where to move these displaced testing assets.

Personnel Management Considerations

Test engineers, who will need to be trained how to use the automation, may have significant technical expertise. The main concern is to induce the test team to accept the new paradigm in which automation is replacing some of their expertise. It is not uncommon to see a reluctance to accept or attempts to discredit the automation.

In the author's experience, the best way to prevent this reluctance is to have the more experienced test engineers help with programming the

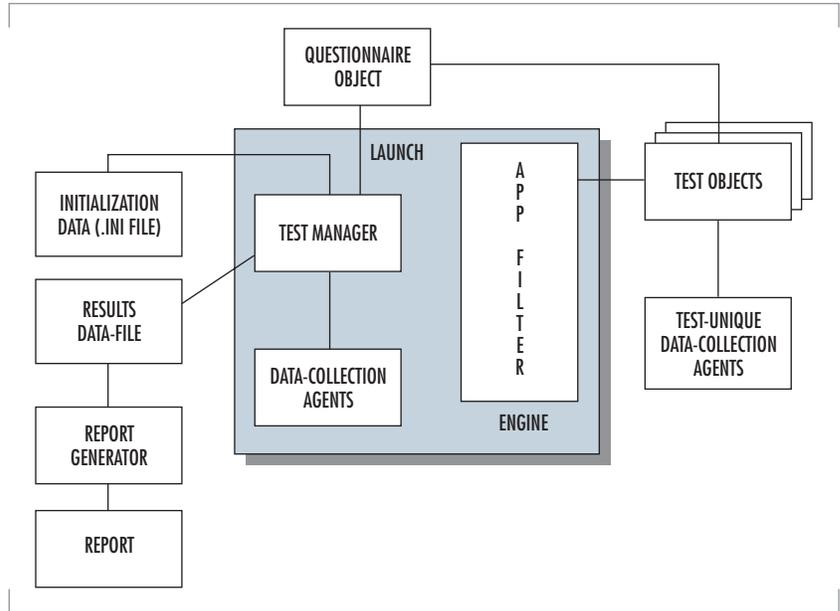


FIGURE 1. Block diagram of testing process.

algorithm-design phase if they are capable. It is important to get them to "take ownership" of the new paradigm. At least a small team of programmers will be required to help with the debugging and enhancing of the automation. The consistency and time savings will pay for these personnel.

The program manager also can redirect some personnel into a quality-assurance role to ensure that the output of the testing is of the required quality. Quality assurance is especially important if the results will be used outside of the test laboratory.

REVIEW

To achieve success in replacing the manual software testing process with an automated testing process, the test engineer must complete the following actions:

1. Document the current testing process.
2. Identify common processes.
3. Complete the following steps in parallel
 - a. Design the automation software.
 - b. Design the automation process.
 - c. Encourage acceptance by the test engineers by inducing them to "take ownership" of the new process.
 - d. If appropriate, consider using an object-design for the automation. Think about using a separate object for each test. Using objects helps when individual tests need to be modified.
 - e. Think about a design that allows new tests and new reports to be added with few or no changes to the underlying data-collection (engine) process.
 - f. When designing the software, think of the "big picture." How can this "machine" be used as part of a bigger system? Perhaps, instead of generating a report, the output could be used as input to a database. In this case, the report capabilities of the database could be used or the defects could be tracked automatically.

CONCLUSION

Replacing manual software testing with automated software testing can yield numerous rewards. A repeatable test process is the major advantage, leading to improved software quality and avoidance of a non-repeatable test. The depth of test coverage also can be increased, and the time requirements can be reduced. The combination of these two factors will improve the quality and cost savings of the software that supports DoD systems compliant with DII COE requirements. This testing methodology could be applied to testing software for government agencies outside DoD, such as the Department of Transportation Federal Aviation Administration and the Department of the Energy, both of which have exacting standards related to safety and security.

ACKNOWLEDGMENTS

This work was sponsored by the Defense Information Systems Agency–Defense Advanced Projects Research Agency (DISA–DARPA) Joint



Jack Chandler

BS in Computer Engineering,
University of New Mexico, May 1991
Current Research: Automation of
repetitive tasks and removal/reduction
of subjectivity; collaboration research.

Program Office. The author thanks the entire test team, especially Pho Le, Steve Bitant, Will Greenway, Todd Webb, Randy Schiffman, MAJ Greg Csehoski and CAPT Stuart Kurkowski.

REFERENCES

1. Dustin, E. 1997. "Process of Introducing Automated Test Tools to a New Project Team," *Proceedings of the Rational User Conference*. URL: <http://www.autotestco.com/html/sld001.htm>
2. Pettichord, B. 1996. "Success with Test Automation," *Proceedings of Quality Week 96*, URL: <http://www.io.com/~wazmo/succpap.htm>

