Destructive Software Testing: A New Testing Paradigm

B. Avinash, R. Ravi, M. Nagendra Rao
Assistant Professor, Dept. of CSE, Mallareddy Institute of Engineering and Technology, Hyderabad.

Abstract- Traditional software testing analyzed to see if a software product meets its specifications. This generally involves testing to illustrate if the software performs all the functions called for in the Software Requirements Specifications (SRS). In contrast, this work-in-progress paper proposes a testing paradigm that does not have this objective. The proposed testing paradigm performs testing to see if a software product exhibits proper behavior when subject to improper usage or improper input. For lack of a more descriptive name and in compliance with similar testing performed on hardware systems, this new paradigm is called "destructive software testing". As presented in this paper, destructive software testing does not replace conventional testing; rather destructive software testing supplements conventional testing (calls for additional testing beyond conventional testing). This paper discusses other uses of the term "destructive software testing" as applied to software systems. Conventional testing techniques are ranked based on applicability to destructive testing. Techniques of incorporating destructive testing requirements into the SRS are proposed, the need and rational for destructive testing is discussed, and ongoing and future work in destructive software testing is outlined.

Keywords- software, engineering, testing, destructive

I. INTRODUCTION

The Conventional software testing has been defined in various ways. Some common definitions are [2] [3][7] [9]:
"Testing is the activity or process which shows or demonstrates that a program or system performs all intended functions correctly"
"Testing is the activity of establishing the necessary „confidence‟ that a program or system does what it is supposed to do, based on the set of requirements that the user has specified"
"Testing is the process of executing a program/system with the intent of finding errors"
"Testing is any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results"
Destructive software testing, as proposed in this paper, does not conform to any of the above definitions. In fact, a lot of destructive testing can be performed without knowledge of the original software requirements of a software product. Some knowledge of the requirements may however sometimes help in developing a good comprehensive destructive testing strategy. These will be discussed later in this paper.

II. BASIC TENETS OF DESTRUCTIVE TESTING

Destructive testing is not a replacement for conventional testing. Rather, destructive testing should be performed in addition to conventional testing. Destructive testing acknowledges the fact those users of software product will sometimes not use the software correctly. Improper or incorrect input data will be supplied, improper or incorrect commands will be typed, improper or incorrect GUI sequences will be applied, and so on.
The old adage of "garbage in, garbage out" is not good enough for high quality, robust, and reliable software. A better adage would be something like "garbage in, proper predictable behavior out". Hence, destructive software testing improves the quality of a software product.
The terminology, "destructive software testing", was chosen in compliance with the corresponding relative concept of "destructive hardware testing", in which hardware systems are destroyed as part of testing. A good example is the testing of automobiles for passenger safety in the event of an automobile accident. The usual practice is to subject the automobile in question to an actual accident in which the automobile is heavily damaged or destroyed.

The term may be a misnomer in the case of software, because the software is not actually destroyed.

Some possible definitions for destructive testing are:
"Testing that assures proper software behavior when the software is subject to improper usage or improper input"
"Testing that attempts to crash a software product"
"Testing that tries to crack or break a software product"
"Testing that checks the robustness of a software product"
"Testing that assures predictable software behavior when the software is subject to incorrect usage or input"

The term "destructive testing" as used in this paper should not be confused with the same term as sometimes used for conventional software testing.
In the case of conventional software testing, the term "destructive testing" has sometimes been used to indicate software that fails conventional testing (see, for example, [2] [10]).

III. APPLICABILITY OF CONVENTIONAL TESTING STRATEGIES TO DESTRUCTIVE SOFTWARE TESTING

Table 1 shows the applicability of several popular conventional testing strategies/concepts, to destructive testing. For more information about each strategy and/or concept, see [3] [7] [8]. The three classifications used in the table are:

A – the strategy cannot be used for destructive testing

B – the strategy can, with modifications, be used for destructive testing

C – the strategy can, without any modifications, be used for destructive testing

1.Incorporation of Destructive Testing into Software Specifications

The requirements for a software system can be written so that it mandates and/or promotes destructive testing. Such requirements are, by definition, non- functional [8]. Functional requirements by nature fall into category A in table 1 (cannot be used for destructive testing). To incorporate mandatory destructive testing into non-functional requirements, clauses similar to the following have to be part of the requirements:

Table 1. Applicability of Conventional Techniques to Destructive Testing.

TESTING STRATEGY/ CONCEPT	APPLICABILITY TOWARDS DESTRUCTIVE TESTING
Black Box Testing	C
Bottom-Up Testing	B
Top-Down Testing	C
Regression Testing	C
Basis Path Testing	B
Interface Testing	C
Security Testing	B
Equivalence Partitioning	C
Test Cases	C
Quality Assurance	C
Quality Control	C
Verification Testing	A
Validation Testing	A
Acceptance Testing	C
Benchmark Testing	A
Boundary Value Testing	C
Loop Testing	C
Defect Testing	B
Stress Testing	C
Alpha Testing	C
Beta Testing	C
Smoke Testing	B
Performance Testing	B
Unit (Module) Testing	B
System (HighOrder) Test	C
Integration Testing	B
Object-Oriented Testing	B

a. The software shall not prematurely, unconditionally or unintentionally terminate as a result of any combination of user keyboard or mouse input.

b. The software shall never accept or process invalid input data.

c. The software shall always produce proper output data regardless of the validity or correctness of input data.

For specific software products, it is important to explicitly define the following and similar terms as used above:

- Proper software behavior as per specifications.
- Improper software behavior according to requirements.
- Improper usage of requirements.
- Improper input data.
- Proper output data.

The author of this paper is currently working with a team to develop requirements specification for an example case study involving a data conversion program. The requirements being developed mandate the use of destructive testing as described above. Subsequent to the completion of the requirements specification and implementation of the software, test cases will be developed for destructive testing of the software.

IV. CONCLUSION

This work-in-progress paper has proposed a software testing paradigm (destructive testing) that deviates from conventional software testing. The goal of conventional software testing is to ensure a software product correctly performs all the functions specified in the requirements specification. In contrast, the goal of destructive testing is to ensure a software product exhibits proper behavior when subject to improper usage or improper input. Ongoing work includes the development of requirements specification that mandates destructive testing of a case study software product. Destructive testing does not replace conventional testing, rather, destructive testing supplements (requires additional testing beyond) conventional testing. In other words, destructive testing is a reflection of the fact that, despite the best of intentions, a software user will sometimes use a software product in an improper manner. Since destructive testing does not replace conventional testing, and it is performed in addition to conventional testing, destructive testing cannot be detrimental. Destructive testing can only be beneficial.

V. REFERENCES

[1]. Drake, Thomas, “Measuring Software Quality:A Case Study”, IEEE Computer, Nov 1996.

[2]. Drake, Thomas, “Testing Software Based Systems: The Final Frontier” Software Technical News, Department of Defense, US Government, Vol 3, No 3, 1999.

[3]. Hetzel, Bill, “The Complete Guide to Software Testing”, John Wiley, 1988.

[4]. Institute of Electrical and Electronic Engineers, IEEE Std 829-1998, IEEE Standard for Software Test Documentation, 1998.

[5]. Institute of Electrical and Electronic Engineers, IEEE Std 1008-1997 (R2003), IEEE Standard for Software Unit Testing, 2003.

[6]. Kan, Stephen, Metrics and Models in Software Quality Engineering, Addison-Wesley, 2003.

[7]. Myers, G. J., The Art of Software Testing, John Wiley, New York, 1979.

[8]. Pressman, R. S, Adaptable Process Model: Software Test Specification, www.rspa.com, 2005.

[9]. Reifer, Don, "Testing Software: Challenges for the Future", Software Technical News, Vol 3, No 2, Department of Defense, US Government, 1999.
[10]. Whittaker, James, How to Break Software, Addison Wesley, Reading MA, 2002.
[11]. Whittaker, James, "Software's Invisible Users," IEEE Software, Vol 18, No 3, pp. 84-88, 2001.
[12]. Whittaker, James, "Software Testing as an Art, a Craft, and a Discipline", Software Technical News, Vol 7, No 2, Department of Defense, US Government, 2004.