Testing Through the Ages

A. T. Dahbura
Motorola Cambridge Research Center
One Kendall Square, Building 200
Cambridge, MA 02139
atd@mcrc.mot.com

Abstract

This paper examines the testing challenges humans have faced through recorded history and discusses why systems and testing methods fail. The paper places in perspective the testing issues which are likely to be encountered over the next several years relative to the gains made thus far.

Keywords

Testing, failures, risks, conformance testing, requirements validation, verification.

“What we anticipate seldom occurs; what we least expect generally happens.”

- Henrietta Temple.

1 Introduction

Throughout history, humans have created increasingly complex systems that put themselves and their environment at risk. In fact, human-made catastrophes appear to have increased in frequency and magnitude with industrialization (Perrow, 1984). Early designers were able to more readily learn from their mistakes, leading to wondrous advances in architecture, transportation, health-care, microelectronics, and so on. Unfortunately, the growing complexity of present-day systems and the critical application areas in which they are used has made it more difficult to produce the technologically-advanced marvels that society relies upon and even expects without also introducing the possibility of catastrophic failures.

F.G. Juenger referred to failures as “resistances”, or obstacles which must be encountered during the design and implementation of any major technological achievement. He observed that these resistances are never overcome, but are merely subdued, “watching in ambush, forever ready to

burst into destruction” (Juenger, 1949). Failures include not only those which affect the intended users and bystanders, but also the producers of the system, where a flaw in a product leads to massive recalls, lost profit, loss of reputation, and so on.

Testing, the process of checking that a system possesses a set of desired properties and/or behaviors, has become an integral part of the invention, production, and operation of systems in order to reduce the risk of catastrophic failures. In many cases the testing methods used are implicit in the design process. Typically, design and test have been treated as discrete components of a system’s life-cycle; only recently have engineers started viewing the two as being intrinsically and inexorably related.

In this paper, we consider different types of complex systems built through history and examine why they, and their design and test methodologies, have succeeded or failed. A characterization scheme is given for systems during their life-cycle and also for the different classes of tests which must be used for different purposes at each stage of the life-cycle. The major challenges for researchers in the field of testing are described.

2 “Normal” Accidents

Charles Perrow has written one of the most insightful treatises on failures and their causes (Perrow, 1984). He refers to what he terms the “interactive complexity” of a system, and argues that accidents are caused by the way failures interact and the way systems are tied together; two or more failures interact in some unexpected way. Perrow refers to this characteristic of the system, which is not due directly to a part or an operator, as a “normal accident” or “system accident”: given the system characteristics, multiple and unexpected interactions of failures are inevitable.

Normal accidents are an inherent property of the system and do not have to be expected or occur frequently. Furthermore, normal accidents are usually incomprehensible by those involved for some critical period of time.

Normal accidents are most prevalent when the system is tightly coupled, that is, processes occur in quick succession. A disturbance propagates quickly and irretrievably, and operator actions or safety systems may even make it worse. Often, portions of the disturbance are masked by other portions.

Perrow characterizes the components of normal accidents as DEPOSE: Design, Equipment, Procedures, Operators (and/or organizations), Supplies/material, and Environment. Usually, several of these factors are contributing causes to an incident becoming an accident.

Complex interactions, those of unfamiliar, unplanned, or unexpected sequences and not visible nor immediately comprehensible, are more common causes of normal accidents than so-called linear interactions, which are those which are expected in a familiar sequence.

For example, on May 25, 1979, an American Airlines DC-10 crashed upon take-off from O’Hare International Airport in Chicago, killing 273 passengers and crew. It was later determined that one of its three engines tore off due to an engine pylon failure, probably because of poor mainte-

nance practices. In spite of this, the DC-10 is designed to fly with only two of its engines operational. Unfortunately, as the engine broke away it severed control cables in the wing, forcing the leading edge slats on one wing to retract. Although the plane is designed to be able to fly in this mode, hydraulic lines in the wing were also severed, preventing appropriate warning indicators in the cockpit to activate; since the pilots were totally unaware of this unexpected state of the aircraft, they were unable to take measures to prevent the tragedy (Perrow, 1984). In this example almost all of the DEPOSE factors (with the possible exception of the environment) played a role in the bizarre sequence of events that led to the disaster.

3 The Phases of a System's Life-Cycle

A system can be thought of as undergoing the following phases from concept to implementation to operation (see Figure 1):

- *requirements phase*: the definition of the functions the system performs and the properties it possesses. Usually, these are described at a very high-level; for example, "the boat must float", "the protocol must not contain deadlocks", "two trains should never collide", etc.
- *specification phases*: these are the various stages of system design, typically starting with a high-level specification, to successively lower-level designs which lead to one which is directly implementable. For instance, in VLSI design, an early specification consists of a block diagram of the architecture. Later phases include RTL-, gate-, and transistor-level specifications, which are then used to produce masks and ultimately, wafers.
- *prototyping phase*: the design is implemented via a model or via simulation, primarily for the purpose of checking that it meets its requirements or conforms to some level of its specification.
- *production phase*: the design is implemented (manufactured) for the purpose of delivery to the end-user. In many cases the implementation is replicated multiple times (mass production).
- *operational phase*: the period during which the system is carrying out the functions it was designed and built to perform.

The requirements phase defines the "what"; the specification phase defines the "how". Ideally, the phases of a systems life-cycle should be discrete; however, it is often the case that some of the phases are merged together or bypassed entirely. This is especially true when modifications are made over time to a system when it is already in its operational phase, and is a major cause of failures, as will be illustrated in the following sections.

4 Classes of Testing

There are several different kinds of testing which are used at the different phases of a system's life-cycle (see Figure 1) and depending on whether a given phase of the system is to be checked with the requirements or with one of the specification levels as the reference.

FIGURE 1. The phases of a system's life-cycle and the different classes of testing.

4.1 Checking that the requirements are met

Requirements validation is the process of checking that a system possesses the desired properties and performs its intended function (for instance, is the airplane capable of flying- see Figure 2).

4.1.1 Requirements validation from a specification

Requirements validation can be performed on a paper design which is close to the actual implementation of the system, or on any later phase. Within this classification scheme, what has traditionally been called *protocol verification* is a special case of requirements validation since, typically, the goal is to detect undesirable properties of the protocol design such as deadlocks and livelocks.

FIGURE 2. An 1876 aircraft design by W.J. Lewis of New York (Moolman, 1980).

4.1.2 Requirements validation from a prototype

The functions of a system can also be validated from its prototype, especially in the case of mechanical systems where the system must interact with its environment in order to validate it with confidence.

Prototypes are often built and validated when some degree of invention or innovation is involved in the system. There are few documented accounts of failed prototyping experiences, perhaps since it is more natural to boast about the successes. However, throughout the course of history, perhaps no experience in design validation from the prototype phase is as celebrated and popularized as the early experiments in aviation. Many of these validation exercises ended in failure, and in many cases the inventors themselves were the ones to test their own contraptions, sometimes with tragic results.

Some clever inventors of the era learned very quickly how to turn failure into success. Hiram S. Maxim, an American who later became a naturalized Briton, was one of the many participants in the frenzied race of the late 1800's to build a flying machine. Maxim's work was unique at the time because he wanted to build aircraft that could support large payloads. He opted for steam power and tubular steel construction. He spent seven years performing simulations in a wind tunnel, and on July 31, 1884 he conducted the first actual test of the machine. The test rig consisted of a rail system which was designed to prevent the machine from lifting more than .75m above the ground; however, the rig broke during the test run, leading to the outcome shown in Figure 3. In spite of the accident, there was evidence that the machine had 'flown', and the inventor claimed that "it was the first time that a powered flying machine had actually lifted itself, and its crew, into the air". In fact, Maxim was later knighted for his accomplishment (Andrews, 1977).

FIGURE 3. H.S. Maxim after the trial of his flying machine (Andrews, 1977).

4.1.3 Requirements validation from a manufactured and/or operational system

Sometimes, the checks performed on a system during the production and operational phases of its life-cycle are functional in flavor, especially in electronic systems. A common form of run-time requirements validation is a form of fault detection called *sanity checking* (Kraft and Toy, 1981). For example, a sanity check which could be used in banking systems is to ensure that an account never has a negative balance. Such checks are a powerful means for detecting software errors.

4.2 Checking that the specifications are met

Design verification is the process of checking that the different stages and/or levels of design or implementation meet a given higher level of specification.

4.2.1 Design verification from a lower-level specification

In many design processes with multiple phases of specification, it is common to check that a lower-level paper design conforms to a higher-level design. In the VLSI design area, this is usually referred to as *hardware verification*. For example, a great deal of effort is spent checking that the gate-level design of a circuit conforms to its functional-level design.

4.2.2 Design verification from a prototype

Once a design is implemented, it is likely to lose a vast degree of observability and controllability of its internal states and signals. In the protocol arena, the problem of verifying an implementation of a system to its specification is known as *protocol conformance testing* (Dahbura, Sabnani, and Uyar, 1990). Note that in conformance testing, the actual function of the system is largely irrelevant; the goal is to ensure that the externally observable behavior of the implementation conforms to its high-level specification such as a finite-state machine description. If the finite-state machine contains errors from the requirements to the specification they may not be detected; however, the observer of the test process may notice that a behavior is abnormal, such as a telephone system which handles calls improperly.

4.2.3 Design verification from a manufactured system

In the *manufacture-time testing* of mass-produced devices such as integrated circuits, the amount of testing which can be economically performed is likely to be limited. Therefore, it is generally assumed that conformance testing is more thorough and is aimed at detecting design faults while manufacture-time testing is for detecting defects in a given instantiation of the implementation. Also, conformance testing is usually performed based on a higher-level specification than manufacture-time testing. For instance, the specification used for conformance testing could be a finite-state machine, while the traditional specification used for manufacture-time testing is at the gate level.

4.2.4 Design verification from an operational system

Testing during the run time of an operational system checks that the instantiation of the system continues to conform to its specification during its operation. Such checks can be performed on- or off-line. If an abnormality is detected, often there are safety systems built in to the system to diagnose, confine, and mask the fault and to reconfigure, recover, and repair the system (Siewiorek and Swarz, 1982). Usually the operational demands placed on a system during its operation limit the rigor of the tests.

4.3 Faulty Requirements

Even an ideal design and implementation of a system cannot be expected to perform properly if they are based on a flawed set of requirements. A requirement can be flawed if:

- it does not anticipate an input (such as a lightning strike) or a sequence of inputs;
- it contains conflicting requirements which are resolved in an imperfect manner in the system design and implementation.

A flawed set of requirements is the Achilles' heel of a system, since even perfect testing throughout the system's life-cycle cannot prevent a tragedy, as will be seen below.

FIGURE 4. The Dale Dyke dam immediately after the disaster of 1864 (Smith, 1972).

5 Case Studies in Failures

Given the phases of a system's life-cycle and the types of tests for each, let us examine case studies of some well-documented failures of complex systems to better understand the reasons for failure and the measures, if any, which could have been taken to prevent them.

5.1 The Bradfield Disaster of 1864

On March 11, 1864, the outlet valves of the Dale Dyke dam, which had been built in 1858 near the town of Sheffield, were closed for the first time to raise the water level in the Bradfield reservoir. That evening, a crack developed and the dam gave way (see Figure 4), setting loose an estimated 200 million gallons of water. Over 250 people were killed. Later analysis determined that the likely cause of the failure was due to the settlement of the heavy clay, shale, and rubble bank around the outlet pipes, causing a gradual erosion and weakening of the interior of the dam. The so-called Bradfield disaster raised the awareness in Great Britain about the considerable social responsibility the dam-builder has (Smith, 1972).

The water level at the time of the breach was below the maximum level provided for in the requirements. The high-level specification may have underestimated the amount and quality of material needed for such a dam, and poor design of the outlet pipes structure could have contributed to the problem; proper design validation could have detected this, although the understanding of dam construction was still limited at that time. On the other hand, it is possible that shoddy materials which did not meet the specification were called for by the detailed design; this could have been questioned by design verification. In the event that the materials that were used were inferior to those called for in the specification, the best hope of detection in this instance was at

manufacture time, since the structure was sealed after that and run-time testing (inspection) would have been nearly impossible.

5.2 Chunnel ghost trains

Earlier this year, it was reported that trains in the “Chunnel”, the tunnel beneath the English Channel, were forced to make emergency stops approximately five times per week due to spurious emergency signals caused by the salt-water mist raised by the trains which short-circuited sensors in the track, thereby mimicking the presence of a train. The effect of salt water on electronic equipment is well-known yet was overlooked in this instance (Wodehouse, 1995).

In this case, it is likely that the requirements did not take into account the salt-water mist as an input to the system (unexpected input). If they did, the next likely scenario is that the sensor which was used did not meet the requirements for salt-water tolerance; the fault could have been introduced in the specification process, at prototype-time, manufacture-time, or at run-time. Appropriate requirements validation, design verification, conformance testing or manufacture-time testing could have detected the problem.

5.3 New York City subway crash

On June 5, 1995, a subway train in New York City crashed into the rear of another train on the Williamsburg Bridge, killing the motorman and injuring 54 people. The cause, as reported by the New York Times, was that the distance between signals, which was set in 1918, is shorter than the stopping distance of today’s longer heavier, and faster trains. The trains were upgraded without a corresponding modification in the control system (Stalzer, 1995).

There are two ways to view this system. If the trains are viewed as external inputs to the control system then the system was given new, unexpected inputs (the newer trains), for which it could not adequately respond. On the other hand, if the trains are to be viewed as being part of the system then part of the system requirements changed over time: more passengers per hour must be transported, although the requirement that there be no collisions remained throughout. As the operational system was transformed over time, the requirements validation process broke down and two conflicting system properties (fast trains, too short stopping distances) were introduced.

5.4 The cruise ship Royal Majesty

On June 10, 1995, the cruise ship Royal Majesty became grounded off the coast of the island of Nantucket, Massachusetts. The National Transportation Safety Board found that a cracked housing of an antenna for the ship’s Global Positioning System (GPS) caused the antenna to fail to receive satellite information regarding the ship’s position. An alarm went off and the autopilot system reverted to dead reckoning (which is less accurate) for its navigation; the alarm was so faint and poorly positioned that no one noticed, and the ship went off course. The accident could have been averted by someone simply looking out the window (Arnold, 1995).

Although one run-time testing mechanism successfully detected the failure of the antenna, the design and/or implementation failed to meet the requirement that the ship’s crew be adequately

FIGURE 5. The Space Shuttle Challenger upon its fateful lift-off (United States, 1986).

warned in the event of a failure. Another run-time test, a sanity check to be performed by the crew to make sure that the ship was not too close to land, was not performed. The inadequate warning system could have been detected by appropriate requirements validation and design verification, including conformance testing to check that the alarm was audible and visible as designed.

5.5 Space Shuttle Challenger disaster

The explosion aboard the Space Shuttle Challenger is one of the most notable technological failures in modern times. On January 18, 1986, flight 51-L of the Challenger began after several launch delays. A mere 73 seconds into flight, a chain reaction of explosions enveloped the Challenger in flames and it broke into several large pieces which plummeted into the Atlantic Ocean. Seven astronauts perished. A Presidential Commission investigating the accident concluded that the cause was the "failure of the pressure seal in the aft field joint of the right Solid Rocket Motor, due to a design unacceptably sensitive to temperature, the effects of reusability, and the reaction of the joint to dynamic loading." (See Figure 5.) Also, the Commission concluded that the decision to launch was flawed since those making the key decisions were unaware of the history of problems concerning the joints; apparently engineers at the company which produced the joints opposed the design but were overruled by management (United States, 1986).

In this unfortunate situation, the specification of the pressure seal (commonly known as an O-ring) did not meet the requirements from the start. In this case, it is likely that requirements validation detected the problem but organizational issues prevented appropriate action. Run-time tests were unable to detect the failure in time to avert disaster.

5.6 The Crash of the DC-10

In the example of a catastrophic failure described in Section 2 of an American Airlines DC-10 on take-off from O'Hare International Airport, one of the early causes of the chain reaction of events was a poor run-time off-line test (in this case, inspection of the engine pylon during maintenance). However, the reaction of the system to the event of the failed pylon contributed in a major way to the crash. The requirements for the DC-10 had provided for the loss of an engine and for the loss of control of the leading edge slats on one wing, but had not explicitly required that the warning system continue to be operational in the event of simultaneous failures. This scenario was not tested (requirements validation) during any of the design or implementation phases of the aircraft.

6 Testing challenges for the future

Thus far, we have discussed the reasons why systems fail and the different classes of tests which can reduce the risk of failure if used appropriately. In some sense, the science of testing is in a race with the science of system design: as system designs introduce new challenges, the techniques for testing them must be able to keep pace.

6.1 The system complexity battle

The most notable challenge faced by testing is the ever-growing system complexity, from micro-processors to nuclear power plants. Designs are accelerated by more powerful description techniques which are able to compress the specification but, unfortunately, merely obscure the internal state of the system, making testing more difficult. In the case of software systems, designers herald specification languages which require fewer lines of code since (assuming that programmers write a constant number of lines of code per day independently of the language) the belief is that the system will be implemented in less time. Furthermore, because commonly used types of software testing claim to be effective in proportion to the size of the code, there is a common, but false, belief that more compressed programs are also more easily testable!

Complex systems which strain the limits of testing bring about the need for new and automated means of modeling the relevant portions of the system to be tested so that the unavoidable trade-offs can be more rationally made. For instance, a telephone could be modeled at the level of its electronic components (with an exponential number of states), or it could simply be modeled in terms of its function: the ability to make and receive a call. Depending on the circumstances, each view has its own set of benefits and risks which need to be exposed and understood.

Finally, it would be fruitful to investigate the impact of different fault models on the effectiveness of testing. In integrated circuit testing, the single stuck-at fault model has been used for years to generate test vectors (Siewiorek and Swarz, 1982). Although VLSI technology has changed radi-

cally over time and single stuck-at faults are not the prevalent fault in digital microcircuits, the stuck-at fault model is still used because:

- the number of single stuck-at faults to be tested for and consequently the number of generated test vectors are proportional to the number of gates in the circuit, which is marginally acceptable, and
- it is generally believed (although there is surprisingly little supporting evidence!) that a test which can detect a high percentage of stuck-at faults will also tend to detect a high percentage of actual faults in the manufactured circuits.

Discovering both enumerable and relevant fault models for different system domains could help to make the test generation process and the test sequence length more manageable.

6.2 Design and test; design for testability

The notion that design and test are inseparable tasks in system development is increasing in popularity but has not yet matured. In particular, the concept of *design for testability (DFT)* has taken hold in the VLSI area but is only nascent in other areas such as software. Researchers have started to develop ideas for DFT in software-based systems, such as improving the controllability and observability and eliminating unneeded state space. Automated tools for assisting implementers in improving these properties in their designs would be extremely useful.

7 A System Success Story: the Great Pyramid

Perhaps the most successful system story of all is also one of the oldest: that of the pyramids built by the Egyptians in ancient times.

August Mencken describes the process of designing and building the Great Pyramid of Gizeh (see Figure 6) and other Egyptian monuments around 2500 B.C. (Mencken, 1963). He points out that probable reasons for their incredible longevity include:

- *simple design and conservative structure*: it is likely that the Egyptians knew that the structures could be made much larger and more complex, although they would be less stable;
- *overengineered design*, perhaps to make up for limited analytical skills and tools;
- *careful measurement*: the Egyptians were almost fanatical about mathematically precise measurements and relationships among them and the heavenly bodies;
- *finite construction window*: pyramids were tombs for the Pharaohs, so their construction span was planned to be about 20 years, the expected duration of the Pharaoh's reign;
- *evolutionary design*: the designs changed very little over time;
- *sufficient manpower* assigned to the task: historians speculate that one of the purposes of the pyramids was to keep large segments of the population employed;
- *no major modifications to the implementation*: once the Pyramids were built, they were left, for the most part, unaltered.

FIGURE 6. The Great Pyramid at Gizeh (Mencken, 1963).

- *luck*: although some parts have been damaged by severe earthquakes and others have settled, there has been no major environmental disaster in the area.

While not all of the pyramids in Egypt have survived, the design principles used are equally applicable today.

8 Conclusions

F.G. Juenger said that “to think in terms of causes, effects, and purposes means to think one-sidedly. To see things in their whole context cannot be learned, no more than one can learn rhythm...Correlations and contexts are noticed only by those minds which think in universal and reverent terms...” (Juenger, 1949).

In this paper, we have argued that, while “normal accidents” have complex causes and are ultimately inevitable, the effective use of the several different classes of tests described earlier can go far in reducing the number of catastrophic system failures which occur. Testing has multiple facets during a system’s life-cycle. It is the professional and even the social responsibility of the system designers to keep the role of testing in its proper context and to use the power of testing to its fullest to effectively reduce risk.

9 References

- Andrews, A. (1977). *Back to the drawing board: the evolution of flying machines*. David and Charles, Ltd., London.
- Arnold, D. (1995). U.S. points to faulty alarm in groundings; ships warned, in *The Boston Globe*, August 11, 1995.
- Dahbura, A.T., Sabnani, K.K., and Uyar, M.U. (1990). Formal methods for generating conformance test sequences, in *Proceedings of the IEEE*, vol. 78, no. 8, pp. 1317-1326.
- Juenger, F.G. (1949). *The failure of technology: perfection without purpose*. H. Regnery Co., Hinsdale, IL.
- Kraft, G.D. and Toy, W.N. (1981). *Microprogrammed control and reliable design of computers*. Prentice-Hall, Inc., New York.
- Mencken, A. (1963). *Designing and building the Great Pyramid*. Privately printed, 1963.
- Moolman, V. (1980). *The road to Kitty Hawk*. Time-Life Books, Alexandria, VA.
- Perrow, C. (1984). *Normal accidents: living with high-risk technology*. Basic Books, New York.
- Siewiorek, D., and Swarz, B. (1982). *The Theory and Practice of Reliable System Design*. Digital Press, New Bedford, MA.
- Smith, N.A.F. (1972). *A history of dams*. Citadel Press, Secaucus, NJ.
- Stalzer, M. (1995) Re: the New York City subway crash, in *The Risks Digest* (P.G. Neumann, ed.), vol. 17, issue 19, June 19, 1995.
- United States- Presidential Commission on the Space Shuttle Challenger Accident (1986). *Report to the President*, vol. 1. The Commission, Washington, D.C.
- Wodehouse, J. (1995) Chunnel has ghost trains, in *The Risks Digest* (P.G. Neumann, ed.), vol. 17, issue 3, April 4, 1995.

10 Biography

Anton T. Dahbura received the BSEE, MSEE, and Ph.D. in Electrical Engineering and Computer Science from the Johns Hopkins University in 1981, 1982, and 1983, respectively.

In 1983 he joined the Computing Systems Research Laboratory at AT&T Bell Laboratories, Murray Hill, NJ, as a Member of Technical Staff. During his tenure at Bell Labs, he conducted research on fault detection and diagnosis algorithms for multiprocessor systems, conformance test sequence generation methodologies for communication protocols, algorithms for memory reconfiguration, and yield enhancement techniques for laser-programmable logic arrays.

The testing method for communications protocols which Dr. Dahbura co-pioneered at AT&T Bell Laboratories, called POSTMAN, has been used extensively by AT&T and others for certification of protocols and product testing, and has become part of a CCITT standard.

In 1990, Dr. Dahbura became the Research Director of the Motorola Cambridge Research Center in Cambridge, Massachusetts. His current responsibilities include management of research activities in the area of parallel and distributed computing systems. His current research interests include dependable and mobile computing, combinatorial optimization algorithms, and applications of the NII.

In 1993, Dr. Dahbura was awarded the IEEE Browder J. Thompson Memorial Prize Award for outstanding paper in any IEEE publication by an author under 30 years of age.

Dr. Dahbura is a Senior Member of the IEEE Computer Society and is a member of the ACM (SIGACT). He is also a member of IFIP Working Group 10.4 (Dependable Computing and Fault Tolerance). In 1988 he was an Invited Lecturer in the Department of Computer Science at Princeton University. Currently, he is a Research Affiliate of MIT's Laboratory for Computer Science.

Since 1991 he has served as an Editor for the IEEE Transactions on Computers. Since 1989 he has served as an Associate Editor for the Journal of Circuits, Systems, and Computers. He has served on the Program Committee of the International Symposium on Fault Tolerant Computing (FTCS) in 1988, 1989, and 1993 and served as Program Co-Chairman of FTCS-24 in 1994. He is the author of over 50 technical papers and holds three U.S. patents.