Regular Paper

Requirements Development, Verification, and Validation Exhibited in Famous Failures

A. Terry Bahill[1,*] and Steven J. Henderson[1,2]

[1]*Systems and Industrial Engineering, University of Arizona, Tucson,* AZ 85721-0020

[2]U.S. *Military Academy, West Point,* NY 10996

Received 11 February 2004; Accepted 31 August 2004, after one or more revisions
Published online in Wiley InterScience (www.interscience.wiley.com).
DOI 10.1002/sys.20017

ABSTRACT

Requirements Development, Requirements Verification, Requirements Validation, System Verification, and System Validation are important systems engineering tasks. This paper describes these tasks and then discusses famous systems where these tasks were done correctly and incorrectly. This paper shows examples of the differences between developing requirements, verifying requirements, validating requirements, verifying a system, and validating a system. Understanding these differences may help increase the probability of success of future system designs. © 2004 Wiley Periodicals, Inc. Syst Eng 8: 1–14, 2005

Key words: design; inspections; case studies

1. INTRODUCTION

Requirements Development, Requirements Verification, Requirements Validation, System Verification and System Validation are important tasks. This paper starts with a definition and explanation of these terms. Then it gives two dozen examples of famous system failures and suggests the mistakes that might have been made. These failures are not discussed in detail: The purpose is not to pinpoint the exact cause of failure, because these systems were all complex and there was no one unique cause of failure. The systems are discussed at a high level. The explanations do not present incontrovertible fact; rather they represent the consensus of many engineers and they are debatable. These explanations are based on many papers, reports, and movies about these failures and discussion of these failures in

*Author to whom all correspondence should be addressed (e-mail: terry@sie.arizona.edu.

Contract grant sponsor: AFOSR/MURI F4962003-1-0377

many classes and seminars since 1997. It is hoped that the reader will be familiar with enough of these systems to be able to apply the concepts of requirements development, verification and validation to some of these systems without an extensive learning curve about the details of the particular systems. When used in classes and seminars, this paper has been given to the students with a blank Table II. The students were asked to read this paper and then Table II was discussed row by row.

2. REQUIREMENTS DEVELOPMENT

A functional requirement should define what, how well, and under what conditions one or more inputs must be converted into one or more outputs at the boundary being considered in order to satisfy the stakeholder needs. Besides functional requirements, there are dozens of other types of requirements [Bahill and Dean, 1999]. Requirements Development includes (1) eliciting, analyzing, validating, and communicating stakeholder needs, (2) transforming customer requirements into derived requirements, (3) allocating requirements to hardware, software, bioware, test, and interface elements, (4) verifying requirements, and (5) validating the set of requirements. There is no implication that these five tasks should be done serially, because, like all systems engineering processes, these tasks should be done with many parallel and iterative loops.

There is a continuum of requirement levels as more and more detail is added. But many systems engineers have been dividing this continuum into two categories: high-level and low-level. High-level requirements are described with words like customer requirements, top-level requirements, system requirements, operational requirements, concept of operations, mission statement, stakeholder needs, stakeholder expectations, constraints, external requirements, and what's. Low-level requirements are described with words like derived requirements, design requirements, technical requirements, product requirements, allocated requirements, internal requirements, and how's. Some of these terms have different nuances, but they are similar. In this paper, we will generally use the terms high-level and low-level requirements, and we will primarily discuss high-level requirements.

3. VERIFICATION AND VALIDATION

Because the terms verification and validation are often confused, let us examine the following definitions:

Verifying requirements: Proving that each requirement has been satisfied. Verification can be done by logical argument, inspection, modeling, simulation, analysis, expert review, test or demonstration.

Validating requirements: Ensuring that (1) the *set* of requirements is correct, complete, and consistent, (2) a model can be created that satisfies the requirements, and (3) a real-world solution can be built and tested to prove that it satisfies the requirements. If Systems Engineering discovers that the customer has requested a perpetual-motion machine, the project should be stopped.

Verifying a system: Building the *system right*: ensuring that the system complies with the system requirements and conforms to its design.

Validating a system: Building the *right system*: making sure that the system does what it is supposed to do in its intended environment. Validation determines the correctness and completeness of the end product, and ensures that the system will satisfy the actual needs of the stakeholders.

There is overlap between system verification and requirements verification. System verification ensures that the system conforms to its design and also complies with the *system requirements.* Requirements verification ensures that the *system requirements* are satisfied and also that the technical, derived, and product requirements are verified. So checking the *system requirements* is common to both of these processes.

There is also overlap between requirements validation and system validation. Validating the top-level system requirements is similar to validating the system, but validating low-level requirements is quite different from validating the system.

Many systems engineers and software engineers use the words verification and validation in the opposite fashion. So, it is necessary to agree on the definitions of verification and validation.

The Verification (VER) and Validation (VAL) process areas in the Capability Maturity Model Integration (CMMI) speak of, respectively, verifying requirements and validating the system. Validation of requirements is covered in Requirements Development (RD) Specific Goal 3 [http://www.sei.cmu.edu/cmmi/; Chrissis, Konrad and Shrum, 2003]. The CMMI does not explicitly discuss system verification.

3.1. Requirements Verification

Each requirement must be verified by logical argument, inspection, modeling, simulation, analysis, expert review, test, or demonstration. Here are some brief dictionary definitions for these terms:

Logical argument: a series of logical deductions

Inspection: to examine carefully and critically, especially for flaws

Modeling: a simplified representation of some aspect of a system

Simulation: execution of a model, usually with a computer program

Analysis: a series of logical deductions using mathematics and models

Expert review: an examination of the requirements by a panel of experts

Test: applying inputs and measuring outputs under controlled conditions (e.g., a laboratory environment)

Demonstration: to show by experiment or practical application (e. g. a flight or road test). Some sources say demonstration is less quantitative than test.

Modeling can be an independent verification technique, but often modeling results are used to support other techniques.

Requirements verification example 1: The probability of receiving an incorrect bit on the telecommunications channel shall be less than 0.001. This requirement can be verified by laboratory tests or demonstration on a real system.

Requirements verification example 2: The probability of loss of life on a manned mission to Mars shall be less than 0.001. This certainly is a reasonable requirement, but it cannot be verified through test. It might be possible to verify this requirement with analysis and simulation.

Requirements verification example 3: The probability of the system being canceled by politicians shall be less than 0.001. Although this may be a good requirement, it cannot be verified with normal engineering test or analysis. It might be possible to verify this requirement with logical arguments.

3.2. Requirements Validation

Validating requirements means ensuring that (1) the *set* of requirements is correct, complete, and consistent, (2) a model that satisfies the requirements can be created, and (3) a real-world solution can be built and tested to prove that it satisfies the requirements. If the requirements specify a system that reduces entropy without expenditure of energy, then the requirements are not valid and the project should be stopped.

Here is an example of an invalid requirements set for an electric water heater controller.

If 70° < Temperature < 100°, then output 3000 Watts.

If 100° < Temperature < 130°, then output 2000 Watts.

If 120° < Temperature < 150°, then output 1000 Watts.

If 150° < Temperature, then output 0 Watts.

This set of requirements is incomplete, what should happen if Temperature < 70°? This set of requirements is inconsistent, what should happen if Temperature = 125°? These requirements are incorrect because units are not given. Are those temperatures in degrees Fahrenheit or Centigrade?

Of course, you could never prove that a requirements set was complete, and perhaps it would be too costly to do so. But we are suggesting that many times, due to the structure of the requirements set, you can look for incompleteness [Davis and Buchanan, 1984].

Detectable requirements-validation defects include (1) incomplete or inconsistent sets of requirements or use cases, (2) requirements that do not trace to top-level requirements [the vision statement or the Concept of Operation (CONOPS)], and (3) test cases that do not trace to scenarios (use cases).

At inspections, the role of Tester should be given an additional responsibility, requirements validation. Tester should read the Vision and CONOPS and specifically look for requirements-validation defects such as these.

3.3. System Verification and Validation

One function of Stonehenge on Salisbury Plain in England might have been to serve as a calendar to indicate the best days to plant crops. This might have been the first calendar, and it suggests the invention of the concept of time. Inspired by a visit to Stonehenge, Bahill built an Autumnal Equinox sunset-sight on the roof of his house in Tucson.

Bahill now wants verification and validation documents for this solar calendar, although he should have worried about this before the hardware was built. This system fails validation. He built the wrong system. The people of England must plant their crops in the early spring. They need a Vernal Equinox detector, not an Autumnal Equinox detector. The ancient ones in Tucson needed a Summer Solstice detector, because all of their rain comes in July and August. System validation requires consideration of the environment that the system will operate in.

In 3000 B.C., the engineers of Stonehenge could have verified the system by marking the sunset every day. The solstices are the farthest north and south (approximately). The equinoxes are about midway between the solstices and are directly east and west. In the 21st century, residents of Tucson could verify the sys-

tem by consulting a calendar or a farmer's almanac and observing the sunset through this sight on the Autumnal Equinox next year. If the sunset is in the sight on the day of the Autumnal Equinox, then the system was built right. When archeologists find Bahill's house 2000 years from now, he wants them to ask, "What do these things do?" and "What *kind* of people built them?"

System-validation artifacts that can be collected at *discrete gates* include white papers, trade studies, phase reviews, life cycle reviews, and red team reviews. These artifacts can be collected in the proposal phase, at the systems requirements review (SRR), at the preliminary design review (PDR), at the critical design review (CDR), and in field tests.

System-validation artifacts that can be collected *continuously* throughout the life cycle include results of modeling and simulation and the number of operational scenarios (use cases) modeled.

Detectable system-validation defects include (1) excessive sensitivity of the model to a particular parameter or requirement, (2) mismatches between the model/simulation and the real system, and (3) bad designs.

At inspections, the role of Tester should be given an additional responsibility, system validation. Tester should read the Vision and CONOPS and specifically look for system-validation artifacts and defects such as these.

A very important aspect of system validation is that it occurs throughout the entire system life cycle. You should not wait for the first prototype before starting validation activities.

3.4. External Verification and Validation

System verification and validation activities should start in the proposal phase. Verification and validation are continuous processes that are done throughout the development life cycle of the system. Therefore, most of theses activities will be internal to the company. However, it is also important to have external verification and validation. This could be done by an independent division or company. External verification and validation should involve system usage by the customer and end user in the system's intended operating environment. This type of external verification and validation would not be done throughout the development cycle. It would not occur until at least a prototype was available for testing. This is one of the reasons the software community emphasizes the importance of developing prototypes early in the development process.

4. FAMOUS FAILURES

We learn from our mistakes. In this section, we look at some famous failures and try to surmise the reason for the failure so that we can ameliorate future mistakes. A fault is a defect, error, or mistake. One or many faults may lead to a failure of a system to perform a required function [www.OneLook.com]. Most well-engineered systems are robust enough to survive one or even two faults. It took three or more faults to cause each failure presented in this paper. System failures are prevented by competent and robust design, oversight, test, redundancy, and independent analysis. In this paper, we are not trying to find the root cause of each failure. Rather we are trying to illustrate mistakes in developing requirements, verifying requirements, validating requirements, verifying a system, and validating a system. Table I shows the failures we will discuss.

HMS Titanic had poor quality control in the manufacture of the wrought iron rivets. In the cold water of April 14, 1912, when the Titanic hit the iceberg, many rivets failed and whole sheets of the hull became unattached. Therefore, verification was bad, because they did not build the ship right. An insufficient number of lifeboats was a requirements development failure. However, the Titanic satisfied the needs of the ship owners and passengers (until it sank), so validation was OK [*Titanic,* 1997]. These conclusions are in Table II.

The **Tacoma Narrows Bridge** was a scaleup of an old design. But the strait where they built it had strong winds: The bridge became unstable in these crosswinds and it collapsed. The film of its collapse is available on the Web: It is well worth watching [Tacoma-1 and Tacoma-2]. The design engineers reused the requirements for an existing bridge, so these requirements were up to the standards of the day. The bridge was built well, so verification was OK. But it was the wrong bridge for that environment, a validation error. [Billah and Scanlan, 1991].

The **Edsel** automobile was a fancy Ford with a distinct vertical grille. The designers were proud of it. The requirements were good and they were verified. But the car didn't sell, because people didn't want it. Previous marketing research for the Thunderbird was successful, but for the Edsel, management ignored marketing. Management produced what management wanted, not what the customers wanted, and they produced the wrong car [Edsel].

In **Vietnam**, our top-level requirement was to contain Communism. This requirement was complete, correct, and feasible. However, we had no exit criteria, and individual bombing runs were being planned at a distance in Washington DC. Our military fought well and bravely, so we fought the war right. But it was the wrong

Table I. Some Famous Failures		
System Name	Year	Putative cause of failure
HMS Titanic	1912	Poor quality control
Tacoma Narrows Bridge	1940	Scaling up an old design
Edsel automobile	1958	Failure to discover customer needs
War in Vietnam	1967-72	No problem statement, Micromanagement
Apollo-13	1970	Poor configuration management
Concorde SST	1976-2003	It was not profitable
IBM PCjr	1983	Failure to discover customer needs
GE refrigerator	1986	Inadequate testing of new technology
Space Shuttle Challenger	1986	Bureaucratic mismanagement, failure to respond to engineers' technical concerns
Chernobyl Nuclear Power Plant	1986	Bad design, Bad risk management
New Coke	1988	Arrogance
A-12 airplane	1980s	Mismanagement, Failure to develop realistic requirements
Hubble Space Telescope	1990	Lack of total system test
SuperConducting SuperCollider	1995	Cost overruns, Failure to maintain public support
Ariane 5 missile	1996	Incorrect reuse of software, Faulty scaling up
UNPROFOR Bosnia Mission	1992-95	No cease fire agreement, Poor coordination
Lewis Spacecraft	1997	Design mistakes
Motorola Iridium System	1999	Misjudged competition, Mispredicted technology
Mars Climate Orbiter	1999	Use of different units
Mars Polar Lander	2000	Failure of middle management
Sept 11 attack on WTT	2001	Failure to anticipate terrorist threat
Space Shuttle Columbia	2002	NASA corporate culture, failure to heed lessons learned
Northeast power outage	2003	Lack of tree trimming

war. We (in Bahill's opinion) should not have been there: bad validation.

John F. Kennedy, in a commencement address at Duke University in 1961, stated the top-level requirements for the Apollo Program: (1) Put a man on the moon (2) and return him safely (3) by the end of the decade. These and their derived requirements were right. The Apollo Program served the needs of Americans: so, validation was OK. But on **Apollo 13**, for the thermostatic switches for the heaters of the oxygen tanks, they changed the operating voltage from 28 to 65 V, but they did not change the voltage specification or test the switches. This was a configuration management failure that should have been detected by verification. On the other hand, perhaps Apollo 13 was a tremendous success and not a failure. The lunar module, the astronauts, the backup systems and the backup crew were robust, so the mission was heroically saved. [*Apollo 13,* 1995].

The **Concorde** Supersonic Transport (SST) was designed and built in the 1960s and 1970s by Britain and France. It flew commercially from 1976 to 2003. The requirements for the airplane were fine and the airplane was built well. But we suggest that it fails validation: because the purpose of a commercial airplane is to make money, and the Concorde did not. [http://www.concordesst.com/]. The Concorde was a success only as a political statement, not as a business system. Once again, these conclusions are not black and white. Indeed one of the reviewers of this paper stated, The Concorde "established a basis of European technical self confidence that permitted Airbus to erode much of the US dominance in this field. Thus, it can reasonably be argued that the Concorde was a successful strategic program."

The **IBM PCjr** was a precursor to modern laptop computers, but it was a financial failure. The keyboard was too small for normal sized fingers. People did not like them and they did not buy them. Modern laptops have normal sized keyboards and PDAs have a stylus. It seems that there is an unwritten requirement that things designed for fingers should be big enough to accommodate fingers. They got the requirements wrong. They build a nice machine with good verification. And the success of present day laptops validates the concept [Chapman, Bahill, and Wymore, 1992: 13].

In 1986, **General Electric** Co. (GE) engineers said they could reduce the part count for their new **refrigerator** by one-third by replacing the reciprocating compressor with a rotary compressor. Furthermore, they said they could make it easier to machine, and thereby cut manufacturing costs, if they used powdered-metal instead of steel and cast iron for two parts. However, powdered-metal parts had failed in their air condition-

Table II Did they do these tasks right?				
Name	Year	RD	VER	VAL
Titanic	1912	No	No	Yes
Tacoma Narrows Bridge	1940	Yes	Yes	No
Edsel automobile	1958	Yes	Yes	No
War in Vietnam	1967-72	Yes	Yes	No
Apollo-13	1970	Yes	No	Yes
Concorde SST	1976-2003	Yes	Yes	No
IBM PCjr	1983	No	Yes	Yes
GE rotary compressor refrigerator	1986	Yes	No	Yes
Space Shuttle Challenger	1986	Yes	No	No
Chernobyl Nuclear Power Plant	1986	Yes	Yes	No
New Coke	1988	Yes	Yes	No
A-12 airplane	1980s	No	No	No
Hubble Space Telescope	1990	Yes	No	Yes
SuperConducting SuperCollider	1995	Yes	Yes	No
Ariane 5 missile	1996	Yes	No	No
UNPROFOR Bosnia Mission	1992-95	No	No	No
Lewis Spacecraft	1997	Yes	Yes	No
Motorola Iridium System	1999	Yes	Yes	No
Mars Climate Orbiter	1999	No	No	No
Mars Polar Lander	2000	Yes	No	Yes
September 11 attack on WTT	2001	No	Yes	Yes
Space Shuttle Columbia	2002	Yes	No	No
Northeast power outage	2003	No	Yes	Yes

ers a decade earlier [Chapman, Bahill, and Wymore, 1992: 19]

Six hundred compressors were "life tested" by running them continuously for 2 months under temperatures and pressures that were supposed to simulate 5 years' actual use. Not a single compressor failed, which was the good news that was passed up the management ladder. However, the technicians testing the compressors noticed that many of the motor windings were discolored from heat, bearing surfaces appeared worn, and the sealed lubricating oil seemed to be breaking down. This bad news was not passed up the management ladder!

By the end of 1986, GE had produced over 1 million of the new compressors. Everyone was ecstatic with the new refrigerators. However, in July of 1987 the first refrigerator failed; quickly thereafter came an avalanche of failures. The engineers could not fix the problem. In December of 1987, GE started buying foreign compressors for the refrigerators. Finally, in the summer of 1988 the engineers made their report. The two powdered-metal parts were wearing excessively, increasing friction, burning up the oil, and causing the compressors to fail. GE management decided to redesign the compressor without the powdered-metal parts. In 1989, they voluntarily replaced over 1 million defective compressors.

The designers who specified the powdered-metal parts made a mistake, but everyone makes mistakes. Systems Engineering is supposed to expose such problems early in the design cycle or at least in the testing phase. This was a verification failure.

The requirements for the **Space Shuttle Challenger** seem to be OK. But the design, manufacturing, testing, and operation were faulty. Therefore, verification was bad [Feynman, 1985; Tufte, 1997]. Bahill thinks that validation was also bad, because putting schoolteachers and Indian art objects in space does not profit the U.S. taxpayer. Low temperatures caused the o-rings to leak, which led to the explosion of Challenger. The air temperature was below the design expectations, and no shuttle had ever been launched in such cold weather. The political decision was to launch in an environment for which the system was not designed, a validation mistake.

The **Chernobyl Nuclear Power Plant** was built according to its design, but it was a bad design: Validation was wrong. Requirements and verification were marginally OK, although they had problems such as poor configuration management evidenced by undocumented crossouts in the operating manual. Human error contributed to the explosion. Coverup and denial for the first 36 hours contributed to the disaster. This is our greatest failure: It killed hundreds of thousands, perhaps millions, of people. Here are references for the U.S. Nuclear Regulatory Commission summary [Chernobyl-1], a general web site with lots of other links [Chernobyl-2], a BBC report [Chernobyl-3], and for

some photos of what Chernobyl looks like today [Chernobyl-4].

The Marketing Department at the **Coca Cola** Company did blind tasting between Coca Cola and Pepsi Cola. Most of the people preferred Pepsi. So, they changed Coke to make it taste like Pepsi. This made Coke more pleasing to the palate, but Coke's branding was a much stronger bond. People who liked Coke would not buy or drink New Coke, and they complained. After 4 months, the company brought back the original under the name of Classic Coke and finally they abandoned New Coke altogether. They did the marketing survey and found the taste that most people preferred, so the requirements development was OK. They manufactured a fine product, so verification was OK. But they did not make what their customers wanted, a validation mistake [http://www.snopes.com/cokelore/newcoke.asp].

The **A-12 Avenger** was to be a carrier-based stealth airplane that would replace the A-6 in the 1990s. This program is a classic example of program mismanagement. Major requirements were inconsistent and were therefore continually changing. Nothing was ever built, so we cannot comment on verification. Validation was bad, because they never figured out what they needed [Fenster, 1999; Stevenson, 2001].

The **Hubble Space Telescope** was built at a cost of around 1 billion dollars. The requirements were right. Looking at the marvelous images we have been getting, in retrospect we can say that this was the right system. But during its development the guidance, navigation, and control (GNC) system, which was on the cutting edge of technology, was running out of money. So they transferred money from Systems Engineering to GNC. As a result, they never did a total system test. When the Space Shuttle Challenger blew up, the launch of the Hubble was delayed for a few years, at a cost of around 1 billion dollars. In that time, no one ever looked through that telescope. It was never tested. They said that the individual components all worked, so surely the total system will work. After they launched it, they found that the telescope was myopic. Astronauts from a space shuttle had to install spectacles on it, at a cost of around 1 billion dollars. [Chapman, Bahill, and Wymore, 1992: 16]

The **SuperConducting SuperCollider** started out as an American Big Science project. Scientists were sure that this system was needed and would serve their needs. But it was a high political risk project. Poor management led to cost overruns and transformed it into a Texas Big Physics project; consequently, it lost political support. The physicists developed the requirements right and the engineers were building a very fine system. But the system was not what the American public, the bill payer, needed. [Moody et al., 1997: 99–100].

The French Ariane 4 missile was successful in launching satellites. However, the French thought that they could make more money if they made this missile larger. So they built the **Ariane 5**. It blew up on its first launch, destroying a billion dollars worth of satellites. The mistakes on the Ariane 5 missile were (1) reuse of software on a scaled-up design, (2) inadequate testing of this reused software, (3) allowing the accelerometer to run for 40 seconds after launch, (4) not flagging as an error the overflow of the 32-bit horizontal-velocity storage register, and (5) allowing a CPU to shutdown if the other CPU was already shutdown. The requirements for the Ariane 5 were similar to those of the Ariane 4: So it was easy to get the requirements right. They needed a missile with a larger payload, and that is what they got: So, that part of validation was OK. However, one danger of scaling up an old design for a bigger system it that you might get a bad design, which is a validation mistake. Their failure to adequately test the scaled-up software was a verification mistake [Kunzig, 1997].

The **United Nations Protection Force** (UNPROFOR) was the UN mission in Bosnia prior to NATO intervention. They had valid requirements (stopping fighting in former Yugoslavia is valid) but these requirements were incomplete because a peacekeeping force requires a cease-fire before keeping the peace. This expanded cease-fire requirement later paved the way for the success of NATO in the same mission. Moreover, the UNPROFOR failed to meet its incomplete requirements because they were a weak force with limited capabilities and poor coordination between countries. UNPROFOR had incomplete requirements, and was the wrong system at the wrong time. This was a partial requirements failure, and a failure of verification and validation [Andreatta, 1997].

The **Lewis Spacecraft** was an Earth-orbiting satellite that was supposed to measure changes in the Earth's land surfaces. But due to a faulty design, it only lasted 3 days in orbit. "The loss of the Lewis Spacecraft was the direct result of an implementation of a technically flawed Safe Mode in the Attitude Control System. This error was made fatal to the spacecraft by the reliance on that unproven Safe Mode by the on orbit operations team and by the failure to adequately monitor spacecraft health and safety during the critical initial mission phase" [*Lewis Spacecraft*, 1998].

Judging by the number of people walking and driving with cellular phones pressed to their ears, at the turn of the 21st century there was an overwhelming need for portable phones and the **Motorola Iridium System** captured the requirements and satisfied this need. The

Motorola phones had some problems such as being heavy and having a time delay, but overall they built a good system, so verification is OK. But their system was analog and digital technology subsequently proved to be far superior. They should have built a digital system. They built the wrong system [http://www.spaceref.com/news/viewnews.html?id=208].

On the **Mars Climate Orbiter**, the prime contractor, Lockheed Martin, used English units for the satellite thrusters while the operator, JPL, used SI units for the model of the thrusters. Therefore, there was a mismatch between the space-based satellite and the ground-based model. Because the solar arrays were asymmetric, the thrusters had to fire often, thereby accumulating error between the satellite and the model. This caused the calculated orbit altitude at Mars to be wrong. Therefore, instead of orbiting, it entered the atmosphere and burned up. But we do not know for sure, because (to save money) tracking data were not collected and fed back to earth. Giving the units of measurement is a fundamental part of requirements development. And they did not state the measurement units correctly: so this was a requirements-development mistake. There was a mismatch between the space-based satellite and the ground-based model: This is a validation error. They did not do the tests that would have revealed this mistake, which is a verification error.

On the **Mars Polar Lander**, "spurious signals were generated when the lander legs were deployed during descent. The spurious signals gave a false indication that the lander had landed, resulting in a premature shutdown of the lander engines and the destruction of the lander when it crashed into the Mars surface. … It is not uncommon for sensors … to produce spurious signals. … During the test of the lander system, the sensors were incorrectly wired due to a design error. As a result, the spurious signals were not identified by the system test, and the system test was not repeated with properly wired touchdown sensors. While the most probable direct cause of the failure is premature engine shutdown, it is important to note that the underlying cause is inadequate software design and systems test" [*Mars Program*, 2000].

The Mars Climate Orbiter and the Mars Polar Lander had half the budget for project management and systems engineering of the previously successful Pathfinder. These projects (including Pathfinder) were some of the first in NASA's revised "Faster, Better, Cheaper" philosophy of the early 1990s. It is important to note that despite early failures, this philosophy has yielded successes and is an accepted practice at NASA [2000].

Some failures are beyond the control of the designers and builders, like the failure of the **World Trade Towers** in New York after the September 11, 2001 terrorist attack. However, some people with 20/20 hindsight say that they (1) missed a fundamental requirement that each building should be able to withstand the crash of a fully loaded airliner and (2) that the FBI did or should have known the dates and details of these terrorist plans. This reinforces the point that many of our opinions are debatable. It also points out the importance of perspective and the problem statement. For example, what system do we want to discuss—the towers or the attack on the towers? For purposes of this paper, we will reflect on the buildings as a system, and assume "they" refers to the World Trade Center designers. Clearly, the buildings conformed to their original design and fulfilled their intended purpose—maximizing premium office space in lower Manhattan—so validation and verification were OK. However, the building's requirements proved incomplete, and failed to include antiterrorism provisions. This is a requirements failure.

NASA learned from the failure of the Space Shuttle Challenger; but they seemed to have forgotten the lessons they learned, and this allowed the failure of the **Space Shuttle Columbia**. At a high level, the Columbia Study Committee said that NASA had a culture that prohibited engineers from critiquing administrative decisions. The NASA culture produced arrogance toward outside advice. After the Challenger failure, they installed an 800 telephone number so workers could anonymously report safety concerns, but over the years that phone number disappeared. At a low level, the original design requirements for the Shuttle Orbiter "precluded foam-shedding by the External Tank." When earlier missions failed to meet this requirement but still survived reentry, NASA treated this as an "in-family [previously observed]" risk and ignored the requirement. But the system still failed to meet its requirements—a verification failure [*Columbia*, 2003; Deal, 2004]. We deem Columbia a system validation failure for the same reasons as the Challenger.

The **Northeast electric power blackout** in August 2003 left millions of people without electricity, in some cases for several days. Some of the causes of this failure were an Ohio electric utility's (1) not trimming trees near a high-voltage transmission line, (2) using software that did not do what it should have done, and (3) disregarding four voluntary standards. Verification and validation were fine, because the system was what the people needed and it worked fine for the 26 years since the last blackout. The big problem was a lack of industry-wide mandatory standards—a requirements failure [http://www.2003blackout.info/].

Table II presents a consensus about these failures. It uses the three CMMI process areas, RD, VER, and VAL. Table II uses the following code:

RD: Requirements Development, Did they get the requirements right?
VER: Requirements Verification, Did they build the system right?
VAL: System Validation, Did they build the right system?
"They" refers to the designers and the builders.

Engineers with detailed knowledge about any one of these failures often disagreed with the consensus of Table II, because they thought about the systems at a much more detailed level than was presented in this paper. Another source of disagreement was caused by the fuzziness in separating high-level requirements from system validation. For example, many people said, "The designers of the Tacoma Narrows Bridge should have been the world's first engineers to write a requirement for testing a bridge in strong crosswinds."

The failures described in Tables I and II are of different types and are from a variety of industries and disciplines. The following list divides them (and a few other famous failures) according to the characteristic that best describes the most pertinent aspect of the systems:

Hardware intensive: GE rotary compressor refrigerator, IBM PCjr, Titanic, Tacoma Narrows Bridge, and Grand Teton Dam
Software intensive: Ariane 5 missile, some telephone outages, computer virus vulnerability, and MS Outlook
Project: A-12, SuperConducting SuperCollider, Space Shuttle Challenger, Space Shuttle Columbia, War in Vietnam, Edsel automobile, Apollo-13, New Coke, UNPROFOR, Lewis Spacecraft, Mars Climate Orbiter, Mars Polar Lander, Three Mile Island, Hubble Space Telescope, Chernobyl Nuclear Power Plant, and Concorde SST
System: Motorola's Iridium system, Western Power Grid 1996, WTT Attacks, and Northeast Power Grid in 1977 and 2003.
Financial: Enron (2003), WorldCom (2003), and Tyco (2003).
More famous failures are discussed in Petroski [1992], Talbott [1993], Dorner [1996], Bar-Yam [2004], and Hessami [2004].

5. SYSTEM AND REQUIREMENTS CLASSIFICATION MODEL

As previously mentioned, because System Verification and Validation are often confused with Requirements Development, it is worth juxtaposing these concepts in a unified model. This model helps further clarify terminology and demonstrates how the concepts of System Verification, System Validation, and Requirements Development interact in the systems engineering design process.

Our model first divides the set of all systems into two subsets: (1) verified and validated systems and (2) unverified or unvalidated systems. The model next divides the set of all system requirements into four subsets (1) valid requirements, (2) incomplete, incorrect or inconsistent requirements, (3) no requirements, and (4) infeasible requirements.

We now explore the union of all possible systems with all possible requirement sets. The result, shown in Table III, is a ***System and Requirements Classification Model*** *(SRCM)*. This model (1) provides a means of categorizing systems in terms of design conformity and requirements satisfaction and (2) provides a way to study requirements not yet satisfied by any system. Each region of the model is discussed below, in an order that best facilitates their understanding.

Region A1: This region denotes the set of systems that have been verified and validated and have valid requirements. It represents systems that conform to their designs and fulfill a valid set of requirements. A properly forged 10 mm wrench designed and used to tighten a 10 mm hex nut is an example of a system in this region. Most commercial systems fall into this category.

Region B1: Region B1 is composed of unverified or unvalidated systems that have valid requirements. These systems have perfectly legitimate designs that, if properly implemented, satisfy a valid set of requirements. Most systems will pass through this region during development. However, some fielded system will be in this region because of poor design realization. Fielded systems in this region are often categorized as error prone or "having bugs." A 10 mm wrench that breaks when tightening a 10 mm hex nut because of poor metal content is an example of a system in this region. These systems are potentially dangerous, be-

Table III. The SRCM Model

Verified and Validated Systems	Unverified or Unvalidated Systems	No System	
A1	B1	C1	Valid Requirements
A2	B2	C2	Incomplete, Incorrect or Inconsistent Requirements
A3	B3	C3	No Requirements
		C4	Infeasible Requirements

cause presumptions about legitimate designs can hide flaws in implementation.

Region A2: This region denotes verified and validated systems that satisfy an incomplete, incorrect, or inconsistent set of requirements. These systems fail because the assigned requirements do not adequately satisfy the stakeholder needs. Adequate satisfaction results when a system meets the needs of a majority of the principal stakeholders (as defined by the chief decision maker). A system in this region would be a properly forged 10 mm wrench that is used to tighten a 10 mm hex bolt, but the bolt fails to tighten because of a previously unnoticed backing/lock nut. In this case, a new system design featuring two tools is required. Another example would be a user's manual that incorrectly instructs a user to apply a properly forged 11 mm wrench to tighten a 10 mm hex nut. The wrench is fulfilling its design (it would have worked for 11mm), adheres to its design (properly forged), but is the wrong tool for the job. The IBM PCjr was a useful and properly functioning computer that was not quite what consumers really wanted. The 2003 Northeast Blackout, with its lack of industry-wide standards, also fits into this region.

Region A3: This region represents systems that are verified (adhere to their designs) and validated, but whose intended designs do not match any significant requirement. A significant requirement is one that is shared by the majority of principal stakeholders. These systems are often described as offering solutions in search of problems. A strange looking (and properly forged) wrench designed to tighten a non-existent nut is an example of a system in this region. Yes, this wrench might eventually satisfy the needs of a customer in a completely foreign context, such as use as a paperweight. However, it does not meet a requirement within the context of its original design (a tool for tightening a nonexistent nut). The weak glue created by 3M stayed in the A3 region for a long time until someone invented Post-it® notes. Their scientists studied the glue carefully, but 3M certainly did not have a requirement for weak glue. In the 1940s IBM, Kodak, General Electric, and RCA were offered the patents for what eventually became the Xerox photocopy machine, but they declined saying that there was no requirement to replace carbon paper.

The types B2 and B3 are unverified equivalents of types A2 and A3, respectively. Not only do these system designs either address nonexisting, incomplete, or incorrect requirements, but the systems also fail to adhere to their designs or fail to satisfy stakeholder needs in

Table IV. SRCM Type for the Famous Failures

Name	Year	RD	VER	VAL	SRCM Type
Most commercial systems		Yes	Yes	Yes	A1
Perpetual motion machine		No	NB*	Yes	C4
Titanic	1912	No	No	Yes	B2
Tacoma Narrows Bridge	1940	Yes	Yes	No	B1
Edsel automobile	1958	Yes	Yes	No	B1
War in Vietnam	1967-72	Yes	Yes	No	B1
Apollo-13	1970	Yes	No	Yes	B1
Concorde SST	1976-2003	Yes	Yes	No	B1
IBM PCjr	1983	No	Yes	Yes	A2
GE rotary compressor refrigerator	1986	Yes	No	Yes	B1
Space Shuttle Challenger	1986	Yes	No	No	B1
Chernobyl Nuclear Power Plant	1986	Yes	Yes	No	B1
New Coke	1988	Yes	Yes	No	B1
A-12 airplane	1980s	No	NB*	No	C2
Hubble Space Telescope	1990	Yes	No	Yes	B1
SuperConducting SuperCollider	1995	Yes	Yes	No	B1
Ariane 5 missile	1996	Yes	No	No	B1
UNPROFOR Bosnia Mission	1992-95	No	No	No	B2
Lewis Spacecraft	1997	Yes	Yes	No	B1
Motorola Iridium System	1999	Yes	Yes	No	B1
Mars Climate Orbiter	1999	No	No	No	B2
Mars Polar Lander	2000	Yes	No	Yes	B1
September 11 attack on WTT	2001	Yes	Yes	Yes	A1
Space Shuttle Columbia	2002	Yes	No	No	B1
Northeast power outage	2003	No	Yes	Yes	A2

*NB means system was not built.

the process. Exemplifying famous failures are listed in Table IV.

Our model also features several other regions of importance. Region C1 represents the set of all valid requirements that have no satisfying system designs. This region represents potential and realizable systems that will satisfy the stakeholder's needs. An affordable and efficient electric car—arguably within our technological grasp—represents one such system. In essence, the art and science of requirements development is the process of pairing requirements from this region to systems in region A1. Entrepreneurship thrives in this region.

Another important region, C4, denotes infeasible requirements. Naturally, there are no overlapping systems here, because technology prevents such systems from being built. A perpetual motion machine represents one such system. However, this category is important because research and development efforts are concerned with expanding the verified/unverified system boundary into this region. We believe requirements will logically and sometimes quietly move from this region into C1 and C2 as technology advances.

Region C2 represent requirements that are incomplete/incorrect and do not have an assigned design. These regions represent systems that could become troublesome inhabitants of the B2 region in the future.

Although system design is not a serial process, because there are many iterative loops, the ideal system design path starts at C3 (or perhaps even C4), goes up the column and then across the row to A1. Other paths are also possible. For example, a previously unnoticed feasible requirement may cause a system to quickly move from C1 directly across to A1. A prototype design process might see a system move from C3 up to C2, then back and forth several times from C2 to A2 until finally reaching the top row (and eventually A1).

Table IV summarizes some of our famous failures and categorizes each failure according to one of the model's ten regions. Two system generalizations were added to demonstrate membership in each important category.

Most of the failures of this report were chosen randomly. They were chosen before this paper was written and before the SCRM model was formulated. The exceptions were the systems added in Table IV that were not in Table II. The systems of this paper do not present an even distribution of types of failure or types of system. Furthermore, the distribution was not intended to reflect real life. Data from the Standish Group [1994] could be used to infer what real world distributions might look like. As an example of our discrepancy from the real world, consider that of our two dozen examples only one, the A-12 airplane, was canceled before any airplanes were built. Whereas the Standish Group Report [1994] said that for software projects that were seriously started, two-thirds were canceled before completion. Figure 1 shows the number of systems of Table IV that fit into each region of Table III.

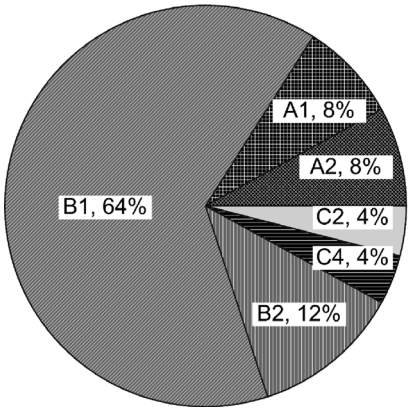


Figure 1. Number of systems from Table IV that are in each cell of the SRCM model.

6. WHAT COULD HAVE BEEN DONE BETTER?

We studied these famous failures and tried to answer the question, "What could they have done better?" Our answers are in Table V. Of course, like all generalizations about complex systems, our answers are not precise. But, nonetheless, they may be helpful. The conclusions in Table V are based on documents with much more detail than was presented in this paper.

7. LESSONS LEARNED

Is it important to develop requirements, verify requirements, validate requirements, verify the system, and validate the system? Yes. This paper has shown examples where failure to do these tasks has led to system failures. Is doing these tasks a necessary and sufficient condition for system success? No. Many systems succeed just by luck; and success depends on doing more than just these five tasks. Is it important to understand the difference between these five tasks? Yes. The CMMI is a collection of industry best practices, and it says that differentiating between these tasks is important. If you can distinguish between these five tasks, you will have a better chance of collecting data to prove that you do these five tasks. This paper has also shown some unique metrics that could be used to prove compliance.

However, there can be controversy about our consensus. Is getting the top-level requirement wrong, a

Table V. Failure Analysis Generalizations	
What could they have done better?	Famous Failure
Gain a better understanding of customer wants, desires and needs	IBM PCjr Edsel Coca Cola UNPROFOR Bosnia Mission
State the top-level problem	Vietnam A-12 airplane
Total system test	Hubble Space Telescope
Disseminate test results more widely	Hubble Space Telescope General Electric refrigerator
Engineering design, testing and verification	HMS Titanic Lewis Spacecraft Northeast power outage Chernobyl Nuclear Power Plant
Configuration management	Apollo 13
Quality control	HMS Titanic
Anticipate unknown problems when scaling up old designs	Tacoma Narrows Bridge Ariane 5 missile
Forecast technology	Motorola Iridium System General Electric refrigerator
Get politics out of business and scientific decisions	Concorde Supersonic Transport Space Shuttle Challenger SuperConducting SuperCollider
Get sufficient money	Chernobyl Nuclear Power Plant Mars Climate Orbiter Mars Polar Lander
Establish good corporate culture	Space Shuttles Challenger and Columbia

system-validation problem or a requirements-development problem? This issue provided the most contention in discussions about these famous failures. It prompted questions such as, "Should they have written a requirement that the Tacoma Narrows Bridge be stable in cross winds? Should they have written a requirement that the Challenger not be launched in cold weather? Should the Russian designers have told their Communist Party bosses that there should be a requirement for a safe design?" For the space shuttles, the top-level requirement was to use a recyclable space vehicle to put people and cargo into orbit. Was this a mistake? If so, what type?

Can we learn other lessons from this paper that will help engineers avoid future failures? Probably not. Such lessons would have to be based on complete detailed failure analyses for each system. Such analyses are usually about 100 pages long.

8. CONCLUSION

In this paper, we have sought to elucidate the frequently confused concepts of requirements development, requirements verification, requirements validation, system verification, and systems validation. After a brief terminology review, we inundated the reader with a casual review of two dozen famous failures. These examples were not offered as detailed failure analyses, but as recognized specimens that demonstrate how shortcomings in requirements development, verification and validation can cause failure either individually, collectively, or in conjunction with other faults. To further distinguish these concepts, we included the system and requirements classification model and several summary views of the failures—failures by discipline, failure generalization, and lessons learned. We hope our approach will promote understanding of terminology and improve understanding and compliance in these five critical systems engineering tasks.

ACKNOWLEDGMENTS

This paper was instigated by a question by Harvey Taipale of Lockheed Martin in Eagan MN in 1997. This paper was supported by Rob Culver of BAE Systems in San Diego and by AFOSR/MURI F4962003-1-0377.

REFERENCES

F. Andreatta, The Bosnian War and the New World Order: Failure and success of international intervention, EU-ISS Occasional Paper 1, October 1997, http://aei.pitt.edu/archive/00000667/.

Apollo 13, Imagine Entertainment and Universal Pictures, Hollywood, 1995.

A.T. Bahill and F.F. Dean, "Discovering system requirements," Handbook of systems engineering and management, A.P. Sage and W.B. Rouse (Editors), Wiley, New York, 1999, pp. 175–220.

Y. Bar-Yam, When systems engineering fails—toward complex systems engineering, International Conference on Systems, Man, and Cybernetics, 2 (2003), 2021–2028.

Y. Billah and B. Scanlan, Resonance, Tacoma Narrows bridge failure, and undergraduate physics textbooks, Am J Phys 59(2) (1991), 118–124, see also http://www.ketchum.org/wind.html.

W.L. Chapman, A.T. Bahill, and W.A. Wymore, Engineering modeling and design, CRC Press, Boca Raton FL, 1992.

Chernobyl-1, http://www.nrc.gov/reading-rm/doc-collections/fact-sheets/fschernobyl.html.

Chernobyl-2, http://www.infoukes.com/history/chornobyl/zuzak/page-07.html.

Chernobyl-3, http://www.chernobyl.co.uk/.

Chernobyl-4, http://www.angelfire.com/extreme4/kiddofspeed/chapter27.html.

M.B. Chrissis, M. Konrad, and S. Shrum, CMMI: Guidelines for process integration and product improvement, Pearson Education, Boston, 2003.

Columbia Accident Investigation Board Report, NASA, Washington, DC, August 2003, pp. 121–123.

R. Davis and B.G. Buchanan, "Meta-level knowledge," Rule-based expert systems, The MYCIN Experiments of the Stanford Heuristic Programming Project, B.G. Buchanan and E. Shortliffe (Editors), Addison-Wesley, Reading, MA, 1984, pp. 507–530.

D.W. Deal, Beyond the widget: Columbia accident lessons affirmed, Air Space Power J XVIII(2), AFRP10-1, pp. 31-50, 2004.

D. Dorner, The logic of failure: Recognizing and avoiding errors in complex situations, Peruses Books, Cambridge, 1996.

Edsel, http://www.failuremag.com/arch_history_edsel.html.

H.L. Fenster, The A-12 legacy, It wasn't an airplane—it was a train wreck, Navy Inst Proc, February 1999.

R.P. Feynman, Surely you are joking, Mr. Feynman, Norton, New York, 1985.

A.G. Hessami, A systems framework for safety and security: the holistic paradigm, Syst Eng 7(2) (2004), 99–112.

R. Kunzig, Europe's dream, Discover 18 (May 1997), 96–103.

Lewis Spacecraft Mission Failure Investigation Board, Final Report, 12, NASA, Washington, DC, February 1998.

Mars Program Independent Assessment Team Summary Report, NASA, Washington, DC, March 14, 2000

J.A. Moody, W.L. Chapman, F.D. Van Voorhees, and A.T. Bahill, Metrics and case studies for evaluating engineering designs, Prentice Hall PTR, Upper Saddle River, NJ, 1997.

NASA Faster Better Cheaper Task Final Report, 2, NASA, Washington, DC, March 2000.

H. Petroski, To engineer is human: The role of failure in successful design, Random House, New York, 1992.

Standish Group International, The CHAOS Report, 1994.

J.P. Stevenson, The $5 billion misunderstanding: The collapse of the Navy's A-12 Stealth Bomber Program, Naval Institute Press, 2001, Annapolis, MD.

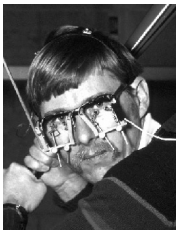
Tacoma1, http://www.enm.bris.ac.uk/research/nonlinear/tacoma/tacoma.html#file.

Tacoma2, http://washington.pacificnorthwestmovies.com/TacomaNarrowsBridgeCollapse.

M. Talbott, Why systems fail (viewed from hindsight), Proc Int Conf Syst Eng (INCOSE), 1993, pp. 721–728.

Titanic, a Lightstorm Entertainment Production, 20th Century Fox and Paramount, Hollywood, 1997.

E.R. Tufte, Visual explanations: Images and quantities, evidence and narrative, Graphics Press, Cheshire, CT, 1997.



Terry Bahill is a Professor of Systems Engineering at the University of Arizona in Tucson. He received his Ph.D. in electrical engineering and computer science from the University of California, Berkeley, in 1975. Bahill has worked with BAE SYSTEMS in San Diego, Hughes Missile Systems in Tucson, Sandia Laboratories in Albuquerque, Lockheed Martin Tactical Defense Systems in Eagan, MN, Boeing Integrated Defense Systems in Kent WA, Idaho National Engineering and Environmental Laboratory in Idaho Falls, and Raytheon Missile Systems in Tucson. For these companies he presented seminars on Systems Engineering, worked on system development teams and helped them describe their Systems Engineering Process. He holds a U.S. patent for the Bat Chooser, a system that computes the Ideal Bat Weight for individual baseball and softball batters. He is Editor of the CRC Press Series on Systems Engineering. He is a Fellow of the Institute of Electrical and Electronics Engineers (IEEE) and of the International Council on Systems Engineering (INCOSE). He is the Founding Chair Emeritus of the INCOSE Fellows Selection Committee. This picture of him is in the Baseball Hall of Fame's exhibition *Baseball as America*.

Steven J. Henderson received an M.S. in Systems Engineering from the University of Arizona in 2003. He is a Captain in the U.S. Army, and is currently serving as an Instructor of Systems Engineering at the U.S. Military Academy at West Point, New York. He graduated from West Point in 1994 with a B.S. in Computer Science. He is currently pursuing a Ph.D. in Systems and Industrial Engineering from the University of Arizona. He is a member of the Phi Kappa Phi honor society, the Institute for Operations Research and Management Sciences (INFORMS), and the American Society of Engineering Educators (ASEE). He was a NASA Faculty Fellow in the summer of 2004.