

Chapter 2

Software Victory - *Exception or Rule?*

Contents

2.1 Software Victory - Exception or Rule?	2-3
2.1.1 Software: The Highest Risk System Component	2-4
2.1.2 Software Disaster Defined	2-5
2.1.3 Long Standing Software Problems	2-6
2.1.3.1 Persistent Software Program Failures	2-7
2.2 Success Vs. Failure	2-8
2.2.1 Where Military Software Acquisition Excels	2-8
2.2.2 Where Military Software Fails	2-9
2.2.3 Obstacles to Improvement	2-10
2.2.4 “No Silver Bullet”	2-12
2.2.5 Defense Science Board Report on Military Software	2-13
2.2.6 Acquiring Defense Software Commercially	2-14
2.3 Why Software Acquisitions Fail	2-14
2.3.1 Management Responsibility	2-14
2.3.2 Technology-Driven Solutions	2-16
2.3.3 Unstable Requirements	2-17
2.3.3.1 Inadequately Stated Requirements	2-17
2.3.3.2 Inadequate User Involvement	2-19
2.3.3.3 Poor Communications	2-20
2.3.4 Software Complexity	2-21
2.3.4.1 Size and Complexity	2-22
2.3.4.1.1 Automated Software Development	2-23
2.3.5 Poor Estimates	2-24
2.3.5.1 Size/Complexity Estimates	2-25
2.3.5.2 Cost/Schedule Estimates	2-25
2.3.5.3 Optimistic Estimates	2-25
2.3.6 Inadequate Software Staffing	2-26
2.3.6.1 Software Labor Shortage	2-26
2.3.6.2 Defense Software Jobs	2-28
2.3.6.3 Labor Shortage Impacts	2-28
2.3.6.4 DoD Hardest Hit by Shortage	2-29
2.3.7 The Domino Effect	2-31
2.4 References	2-32

2.1 Software Victory - Exception or Rule?

Our warfighters are staking the security of the Free World on their arsenal of software-intensive weapons and information systems. The problems caused by poor acquisition management — leading to project delays, cancellations, unreliable software, and huge cost overruns — are as serious a threat to our national security as cyber terrorism or the proliferation of weapons of mass destruction by our enemies. As the Chairman of the Joint Chiefs of Staff warns in **Joint Vision 2010**,

“This era will be one of accelerating technological change. Critical advances will have enormous impact on all military forces. Successful adaptation of new and improved technologies may provide great increases in specific capabilities. Conversely, failure to understand and adapt could lead today’s militaries into premature obsolescence and greatly increase the risks that such forces will be incapable of effective operations against forces with high technology.” [JCS96]

InformationWeek, laments that, “Uncle Sam is the single largest purchaser of computers and computer-related equipment in the world, but he’s been a terrible shopper and manager.” [CONE98²] The problems go way beyond just the well-publicized debacles like the Federal Aviation Administration’s Advanced Automation System (AAS). According to Senate investigator Don Mullinax,

“There are so many horror stories. We see programs where \$40 million gets spent, or \$90 million, and then they have to start over because they didn’t know what they wanted to do.” —Don Mullinax [MULLINAX98]

A 1994 report by the Senate Committee on Government Affairs, *Computer Chaos*, described a grim and widespread situation. “The Federal Government continues to operate old, obsolete computer systems while it has wasted billions of dollars in failed computer modernization efforts.” The report blamed the failures on “poor management, inadequate planning, and an acquisition process that is too cumbersome.” [CONE98¹]

DoD, in particular, has had a distressing history of procuring elaborate, high-tech software-intensive systems that do not work, and cannot be relied upon, modified, or maintained. Many of these over-budget, overdue programs have been canceled after reaching full-scale production — with millions of dollars wasted and not a single system reaching the warfighter. In 1985, *Business Week* summarized situation.

“When the Pentagon decides to build a complex new weapon these days, it often seems to run into disaster. The promise of advanced technology seduces designers and eager contractors into taking big risks with the public’s money. Frequently, these elaborate projects end up hamstrung by technical errors, management miscalculations, or congressional interference. The result is weapons that are grossly overpriced — or don’t work.” [BUSWEEK85]

Over a decade later, problems persist. On April 27, 1995 Frank C. Conahan, senior defense and international affairs advisor to the U.S. Comptroller General, testified before the U.S. House of Representatives Committee on the Budget. He explained that,

“Over the years, we have reported on the persistent problems that have plagued weapons acquisition. Many new weapons cost more, are less capable than anticipated, and experience schedule delays. These problems are typical of DoD’s history of inadequate requirements determinations for weapon systems; projecting unrealistic cost, schedule, and performance estimates; developing and producing weapons concurrently; and committing weapon systems to production before adequate testing has been completed.” — Frank C. Conahan [CONAHAN95]

2.1.1 Software: The Highest Risk System Component

When a major procurement program turns into a fiasco, when costs soar, deliveries fall behind schedule, and performance is compromised, the problem can often be traced to one high-risk component - the software! With virtually every major acquisition failure, the software component can be isolated as a prime contributor to the problem. In December 1990, a series of articles ran in The Washington Post explaining that,

“Software problems have caused major delays of weapons systems, created malfunctioning aircraft, and cost the Defense Department billions of dollars in unanticipated costs. Officials acknowledge that virtually every troubled weapon system, from the electronics of the B-1B bomber to satellite tracking systems, has been affected with software problems. Even straightforward record-keeping systems can get bogged down; last year the Navy canceled a software accounting project nine years in the making after its cost quadrupled to \$230 million.” — Evelyn Richards [RICHARDS90¹]

In the same series, Colonel Joseph Greene, Jr. (USAF), head of a Pentagon software research effort, was interviewed about a study he conducted of 82 large military acquisition programs. Of those, Greene found that programs developing large amounts of software ran 20 months behind schedule — three times longer than non-software-intensive programs. He calculated that those delays cost DoD one tenth of its FY90 \$100 billion research and procurement budget. Greene explained that, *“[t]he department is paying a huge penalty for not dealing with its software problems. The penalty is not just late software — it is degraded war-fighting capability.”* [GREENE90] In another study *The Washington Post* cited, 3/4 of 55 aerospace and defense contractors ran their software programs in an ad hoc, chaotic manner. [RICHARDS90²] The good news is that by June on 1999, that number had decreased to 29%.

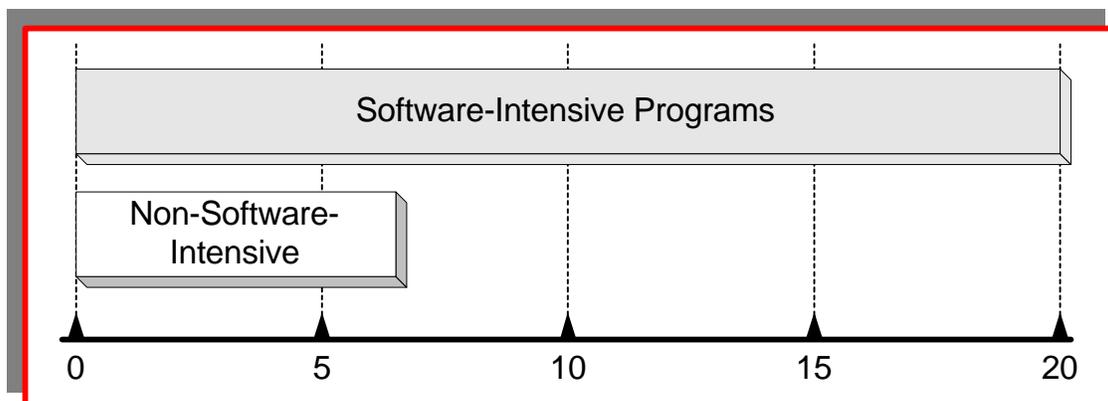


Figure 2-1. Average Schedule Delays (in Months) for Large Military Programs

Previously taken for granted, software is now recognized as the highest risk system component in virtually every major defense acquisition. A Defense Acquisition Board report stated that the estimated 1.55 million lines-of-code to be built for the F-22 Raptor — the largest software task ever undertaken on an attack/fighter program — represents the most significant risk to successful deployment. Interviewed by the General Accounting Office (GAO), F-22 program managers explained that avionics software integration was the most formidable task for their contractors. [GAO95]

2.1.2 Software Disaster Defined

In his book, Death March: The Complete Software Developer's Guide to Surviving "Mission Impossible" Projects, software guru, Ed Yourdon, defines a software disaster, *death march* program as the follows.

"A death march project is one for which an unbiased, objective risk assessment (which includes an assessment of technical risks, legal risks, political risks, etc.) determines the likelihood of failure is >50%." — Edward Yourdon [YOURDON97]

Regrettably, Yourdon concludes that, "*Death march projects are the norm, not the exception.*" Depending on the development organization's cultural idiosyncrasies, Yourdon further categorizes *death march* programs as "mission impossible," "ugly," "suicidal," and "Kamikaze." [YOURDON97]

Stephen Flowers explains in his book, Software Failure: Management Failure, that there are varying degrees of software acquisition failure, including "*flawed but usable*," "*totally unusable*," "*unused*," and "*absolute disaster*." He says a program can be called a *failure* if it falls into the one (or more) of the following categories:

- **Never used** - On implementation, it does not perform as originally intended, or it is so *user-hostile* it is rejected by users.
- **Cost exceeds benefits** - The cost of development exceeds any benefits the system may bring during its useful life.
- **Not completed** - Due to problems with system complexity, program management, or program longevity (where the development is no longer relevant due to advances in technology), the system is abandoned before it is completed. [FLOWERS96]

NOTE: Throughout these Guidelines, a distinction is made among the terms program, application, and project. The following definitions apply:

- **Program: A major acquisition or software development.**
- **Project: A less than major software development or a smaller subsystem development.**
- **Application: Often referred to as "software" or "computer program," the code and documentation component of a computer system.**

2.1.3 Long Standing Software Problems

“Progress, far from consisting in change, depends on retentiveness...Those who cannot remember the past are condemned to repeat it.” — George Santayana [SANTAYANA05]

Although the software industry is reaching its 50-year mark, most of the same problems that have plagued the acquisition of software persist. DoD is not alone in its inability to acquire successful software-intensive systems. Software acquisition failures are also commonplace in the private sector. Industry surveys and years of data collected by software statistical experts, such as Capers Jones, Paul Strassmann, Larry Putnum, and Howard Rubin, indicate that the average software program is:

- 6 to 12 months behind schedule, and
- 50 to 100% over budget. [YOURDON97]

For example, the Standish Group is a Massachusetts-based research firm who has conducted extensive surveys on the software industry. Their findings are listed in Table 2-1. Standish Group defines a software acquisition program failure as one that is abandoned before delivery or that is totally unusable upon completion. [JAMES97]

Multiyear Software Industry Studies Survey	
Program Success	Result
% Successful	27%
% Cost overruns	33%
% Schedule overruns	33%
% Total program failures	40%

Table 2-1. Standish Group Software Industry Survey Results

In a report prepared for Rome Laboratory’s **Data & Analysis Center for Software (DACS)**, McGibbon states that production in the average U.S. software company is poor. [McGIBBON96] The results of the McGibbon survey are listed in Table 2-2.

Extensive Literature Review Survey	
Success Factor	Result
% Software programs cancelled	25%
% Latent defects in delivered software	15%
% Time spent on rework	30% - 40%
% Schedule overruns	50%

Table 2-2. DACS Software Industry Survey Results [MCGIBBON96]

“Software’s Chronic Crisis,” a *Scientific American* article by Wyatt Gibbs cites the results of an IBM study of 24 leading companies developing large, distributed software systems is summarized on Table 2-3. Gibbs says large-scale software development failures do not function as intended or are not used at all. In addition, software is still handcrafted by artisans using techniques they

can neither measure nor consistently repeat. Table 2-4 lists the average performance on large-scale software development programs industry-wide. [GIBBS94]

24 Distributed Software Developers	
Program Success Factor	Survey Result
% Cost overruns	55%
% Schedule overruns	68%
% Software system rework	88%

Table 2-3. IBM Software Industry Survey Results [GIBBS94]

Large-Scale Software Development Programs	
Survey Factor	Survey Result
% Programs cancelled	33%
% Schedule overruns	50%
% Operational failures	75%

Table 2-4. *Scientific American* Software Industry Survey Results [GIBBS94]

Finally, Table 2-5 lists several examples of non-military software program failures.

YEAR	PROJECT	RESULTS
1980's	International Telegraph & Telephone (ITT) - 4 <i>switching systems</i>	<ul style="list-style-type: none"> • 40,000 function point system • \$500 million lost • Cancelled
1987	California Department of Motor Vehicles, Automated Vehicle/Drivers License System	<ul style="list-style-type: none"> • 3 (5,000 function point size) switches • \$30 million lost • Cancelled
1989	State of Washington - Automated Social Service Caseworker System	<ul style="list-style-type: none"> • 7 years to build • Failed to meet user needs • \$20 million lost • Cancelled
1992	American Airlines - Flight Booking System	<ul style="list-style-type: none"> • \$165 million lost • Cancelled

Table 2-5. Major Non-Military Software Failures [GIBBS94]

2.1.3.1 Persistent Software Program Failures

In the preface to his book, Flowers explains why software managers fail to learn from the mistakes of failed programs.

“In researching this book, time and again I came across the strange collusion that exists between the buyers and sellers of information systems. When things go wrong with an information system development, it will almost always result in both sides being eager to bury the facts of the case. Confidentiality agreements, non-disclosure contracts, and undisclosed out-of-court settlements are all ways of trying to keep the lid on what is a seething cauldron of failure. And the most likely result of this industrial amnesia? More of the same.” — Stephen Flowers [FLOWERS96]

The reason we keep making the same mistakes over and over is that we refuse to listen and see. And those things which we do learn we do not act upon! For us to conquer the software war, we must win the management battle. If we do not reengineer the software acquisition task, our programs are doomed to join the ranks of the software norm — *program failure!* The purpose of these Guidelines is to equip you with the ability to avoid many problems and recognize others while they are still solvable. Once you understand what these problems are and why they occur, you will be equipped to do something about containing or eliminating them in your program.

“It’s fine to celebrate success, but it is more important to heed the lessons of failure.” — Bill Gates [GATES95]

2.2 Success Vs. Failure

If all our acquisitions were as bad as some of them, we could not be the world leader that we are. Even so, the road to improvement requires self examination, of both our successes and failures. Both can give us valuable information about what we are doing right, and what we are doing wrong. Too much emphasis on the good blinds us to situations and practices that will eventually bring our downfall. Focusing exclusively on the darker side can overwhelm our determination to improve and hide those things bring success. The system is worth saving, but it will require an in-depth consideration of where and why we have success and failure in military acquisitions.

2.2.1 Where Military Software Acquisition Excels

Capers Jones identified several areas where military software acquisition has *best in class* attributes when compared to commercial software, as illustrated in Table 2-6. Much of this has resulted from the efforts of forward thinking project managers and others who are concerned with the long range future.

Factors Where Military Software Acquisition Excels
It leads in process assessment and process analysis
It leads in applications larger than 100,000 function points
It is among the best in reusability research
It is among the best in CASE research
It is the world leader in Ada language research
It is a world leader in configuration control
It is a world leader in requirements traceability
It is among the best in quality control for weapons systems
It has the highest frequency of cost estimating tool usage

Table 2-6. Why Military Software Excels [JONES95]

The factors to which Jones attributes the success of military and commercial software programs are listed in Table 2-7.

Military Software Acquisition Success Factors	Commercial Software Acquisition Success Factors
Contract was let without litigation	Product achieves significant market share
Project adheres to relevant commercial standards	Product is profitable
Project adheres to best commercial practices	Product prevails in any litigation
Product is highly reliable with excellent quality	Product protects unique features
Project conforms to all requirements	Product leads to follow-on business
Requirements are stable within 15%	Customer support is good to excellent
Schedules are predictable within 10%	User satisfaction is good to excellent
Costs are predictable within 10%	Feature set is better than competitors
Project passes critical design review (CDR)	Time to market is better than competitors
Product actually deployed and used	Quality levels are good to excellent

Table 2-7. Military and Commercial Software Success Factors [JONES95]

2.2.2 Where Military Software Fails

Capers Jones has also identified why military software acquisition program failures outnumber commercial software failures, especially in logistics support and command and control (C2) applications. In the commercial world, software product failure often leads to corporate bankruptcy, where industry averages approach, or exceed, a 50% product failure rate. In any given market niche, Jones explains that a rule of thumb is,

- 10% of commercial software products are very successful,
- 20% are mildly successful,
- 40% are marginal, and
- 30% are failures.

The factors Jones identifies as leading to military and commercial software failures are listed in Table 2-8.

Military Software Failure Factors	Commercial Software Failure Factors
Contract is challenged in court	Product fails to achieve market share
Project adheres to poor civilian practices	Product is readily imitated
Product is very unreliable and of poor quality	Product loses significant litigation
Product fails to meet all requirements	Product generates no ancillary business
Requirements are out of control	Customer support is poor to marginal
Schedules are out of control	User satisfaction is poor to marginal
Costs are out of control	Feature set lags competitors
Project fails critical design review (CDR)	Time to market lags competitors
Product not used or not deployed	Quality levels are poor to marginal

Table 2-8. Factors Leading to Military and Commercial Software Acquisition Failure [JONES95]

Jones claims the military software world lags behind the civilian software world by quite a few years. The factors creating this discrepancy are listed in Table 2-9.

Factors Where Military Software Lags Behind
It lags in the adoption of fundamental metrics
It lags in the productivity measurement technology
It excels all other industries in the production of large documents
Its schedules are longer than any other kind of software project
Its productivity is lower than for any other industry
Its contracts for software have the highest rates of challenges and litigation
Its contractors rank first in layoffs and downsizing
Its contractors lag in staff benefits and compensation
Its contractors lag in training and education of technical staff
Its contractors lag in training of project managers
Its contract software has the highest growth of creeping user requirements
Its contracts associated with SEI maturity levels are much less effective than civilian performance-based contracts

Table 2-9. Factors Where Military Software Lags Behind Commercial Software [JONES95]

2.2.3 Obstacles to Improvement

Improvement is always blocked by obstacles. In his book, Software Failure: Management Failure, Stephen Flowers lists the obstacles to Defense software acquisition success.

- **Technology-driven.** Tendency for acquisitions to be technology-driven, rather than customer-driven. There is no fear that the customer (the warfighter) will take their business elsewhere.
- **Low-cost solutions not sought.** Solutions making use of leading-edge technologies results in high-cost, high-risk approaches rather than low-cost, proven solutions.
- **Short-term tenure of DoD acquisition managers.** Lack of management continuity needed to oversee large-scale, long-term software developments has adverse affects on program focus and progress.

- **Changing program priorities.** Changes in federal policy and funding priorities can negatively impact development program schedule and cost.
- **Imposition of external deadlines.** National security needs impose deadlines by which software products must be operational.
- **Bureaucratic decision-making process.** The level of government oversight reflects the need to ensure the highest level of accountability for the use of public funds. Financial integrity and federal funding and budgeting procedures have created organizational structures unsuited for effective software program management.
- **High level public access and oversight.** Compared to commercial software acquisition programs, which are usually shrouded in secrecy, defense program information, as well as outcomes, are open to public scrutiny. [FLOWERS96]

Hinton suggests that the obstacles to implementing commercial best practices in Defense software-intensive acquisitions include the following.

- **Definition of success.** The definition of software-intensive program success is more complicated in DoD than in the commercial world. Success is defined as getting DoD and the Congress to fund the acquisition program on an annual basis. Optimistic assessments of system performance and cost help ensure this kind of success; realistic risk assessments of unknowns do not. If problems arise in software development, there is not nearly the risk of failure that a commercial product faces. By the time a software-intensive system enters production, the point of sale to the customer has already occurred.
- **Acquisition funding instability.** As a Defense acquisition program proceeds, program success is measured in terms of the funding it receives every year within the budget process. Failure can mean anything from a funding cut to cancellation. In addition, DoD acquisition programs do not receive the corporate support commercial programs do. There is feeding frenzy-like competition among and within the Services for their piece of the acquisition budget pie.
- **Overly optimistic estimates.** DoD acquisition programs are scrutinized by Service executives, OSD, independent cost estimating and testing agencies, audit agencies, and the Congress. With this competition and oversight, reporting software acquisition risks or full-blown problems make programs vulnerable to criticism and possible loss of funding. These pressures encourage overly optimistic cost, performance, and schedule estimates. This contrasts with commercial software-intensive acquisitions, where once a program is launched it receives full corporate support and resources. Commercial program estimates are kept realistic because failure to succeed may affect the firm's bottom-line and future.
- **Risk management.** Problems or indications that overly optimistic estimates are decaying do not help sustain a software-intensive acquisition in subsequent years. Thus, their admission is implicitly discouraged. Likewise, good software system test results can help a program, whereas negative test results are equated with failure. Not knowing how the system performs presents a safer course of action, and testing is delayed until late in system development. *Estimates* of system performance (realistic or not) are safer than bad news.
- **Technology-based solutions.** Risks in the form of ambitious software technology advancements and tight cost and schedule estimates are accepted as necessary for a successful program launch. There is little incentive to admit to high risks until it is absolutely necessary, because that may doom the program. As long as estimates are accepted by DoD and the Congress and the program is funded, the program manager is successful. [HINTON98]

2.2.4 “No Silver Bullet”

“Of all the monsters that fill the nightmares of our folklore, none terrify more than the werewolves, because they transform unexpectedly from the familiar into horrors. For these, one seeks bullets of silver that can magically lay them to rest. The familiar software project, at least as seen by the nontechnical manager, has something of this character; it is usually innocent and straightforward, but is capable of becoming a monster of missed schedules, blown budgets, and flawed products.”
— Frederick P. Brooks, Jr. [BROOKS87]

The April 1987 issue of *Computer* included the most famous, and oft quoted, paper on the software crisis ever published, “No Silver Bullet: Essence and Accidents of Software Engineering,” by software pioneer Frederick P. Brooks, Jr. In it, Brooks compared our software problems to werewolves. Like werewolves, software programs are often unexpectedly transformed from *commonplace* to untamable monsters. Without warning, a normal, routine software development has the capability of becoming a disaster. While a Silver Bullet has the power to slay a werewolf, Brooks claims no Silver Bullet exists that can magically slay our software problems.

Brooks saw two major obstacles to improving the way we build software. One is the *essence* of software — the difficulties inherent in the software beast itself. The other is the *accidents* — those difficulties found in the production of software that are not inherent. Of the essence, Brooks took the position that *software is hard* and always will be because it has an inherent and necessary complexity. Brooks explained that,

“[s]oftware entities are more complex...than perhaps any other human construct...Software systems have orders-of-magnitude more states than computers do...[and] [t]he complexity of software is an essential property.” — Frederick P. Brooks, Jr. [BROOKS87]

This complexity does not lend itself to the simplification techniques found in other disciplines. For example, the field of mathematics uses simplified models of complex problems as analytical tools. The essence of software is that it achieves the solution of a complex problem by compounding its complexity (i.e., the algorithms defining the solution are more complicated than the real-world problems they solve.) [GLASS91]

Brooks also analyzed major breakthroughs, which have increased productivity and improved software quality over the years. Advances, such as high-order languages, faster processing times, and computer-aided software engineering environment (CASE) tools, have achieved quantum leaps in dealing with the accidents. However, he said that it is doubtful technological advances of any magnitude will solve our chronic problems. The promises of Silver Bullets that will yield spectacular progress in software development (common occurrences in the hardware arena) are not to be believed.

Brooks’ recommended solutions to the software dilemma are rather mundane compared to his description of the problem. He tells us *“a disciplined, consistent effort to develop, propagate, and exploit the following suggestions should yield an order of magnitude improvement.”*

- **Buy software, rather than build it.** “Every day this becomes easier, as more and more vendors offer more and better software products for a dizzying variety of applications.”
- **Grow software; don’t build it.** Develop software incrementally and refine requirements through prototyping. Partial solutions are easier to correct and modify than a full-blown, finished product that does not perform as envisioned.
- **Employ and cultivate the best and the brightest.** “Sound methodology can empower and liberate the creative mind; it cannot inflame or inspire the drudge. Great designs come from great designers!” [BROOKS87]

2.2.5 Defense Science Board Report on Military Software

A final report was released in 1987 by the Defense Science Board (DSB) Task Force on Military Software. The Task Force was chaired by Frederick Brooks and manned by some of the most astute experts in the field. It pulled no punches in waging a frontal attack on DoD’s on-going software troubles when it stated:

- *“Many previous studies have provided an abundance of valid conclusions and detailed recommendations. Most remain unimplemented. If the military software problem is real, it is not perceived as urgent.”*
- *“We do not see any single technological development in the next decade that promises ten-fold improvement in software productivity, reliability, and timeliness.”*
- *“Few fields have so large a gap between best current practice and average current practice.”*
- *“The Task Force is convinced that today’s major problems with military software development are not technical problems, but management problems.”* [DSB87]

The report addressed the institutions governing military software development, and the obstacles encountered when transitioning technology and modern management practices to a new engineering discipline. The report’s recommendations for improving software development are summarized as follows.

- DoD should assume software requirements will be met with COTS subsystems and components until it is proved they are unique [requirements].
- DoD should develop metrics and measuring techniques for software quality and completeness, and incorporate these routinely in contract.
- DoD should examine and revise regulations to approach modern commercial practice insofar as practicable and appropriate.

CAUTION! Only adopt the commercial practices that enhance the successful program attributes discussed throughout these Guidelines. Avoid all others!

- DoD should mandate iterative setting of specifications, rapid prototyping of specified systems, and incremental development.
- DoD should mandate the use of risk management techniques in software acquisition.
- DoD should develop economic incentives for contractors to offer modules for reuse and to buy modules rather than building new ones.
- DoD should enhance education for software personnel.

The thrust of the DSB report is summarized in the following statement.

“We call for no new initiatives in the development of technology, some modest shift of focus in technology efforts under way, but major re-examination and change of attitudes, policies, and practices concerning software acquisition.” [DSB87]

2.2.6 Acquiring Defense Software Commercially

The 1994 Defense Science Board Report, *Acquiring Defense Software Commercially*, makes an important observation about trends (or lack thereof) in DoD software acquisition. It states that, “[d]espite the increased emphasis given to software issues by the DoD...the majority of the recommendations resulting from these studies have not been implemented.” [DSB94] The report’s recommendations germane to this discussion include the following:

- Establish mechanisms to allow both current ability to perform and past performance as key factors in source selection;
- Define software architectures to enable rapid changes and reuse;
- Facilitate early systems engineering and iterative development;
- Require program managers to stay with programs at least through beta testing to maintain continuity and understanding of original requirement nuances.

2.3 Why Software Acquisitions Fail

2.3.1 Management Responsibility

“When I entered the workforce in 1979, the Peter Principle (the concept by which capable workers were promoted until they reached their level of incompetence) described management pretty well...Lately, however, the Peter Principle has given way to the ‘Dilbert Principle.’ The basic concept of the Dilbert Principle is that the most ineffective workers are systematically moved to the place where they can do the least damage: management.” — Scott Adams [ADAMS96]

As illustrated above, industry and government experts have arrived at the same conclusion: DoD’s inability to build reliable, economical software is due primarily to *poor management practice*. Management problems are *people problems*. Success or failure depends on the experience, skills, and ability of acquisition and development teams to critically evaluate and improve process and product quality. Poorly trained managers, or a misunderstood and immature acquisition or engineering process leads to unpredictable costs, schedules, and product quality. According to Senate staffer, Bill Greenwalt,

“Management is the most difficult component of the problems Federal Agencies face. We have a long way to go.” [GREENWALT98]

Blum sees the software development process from three perspectives. He describes software design as looking forward, software quality assurance as looking backward, and management as looking downward. Although not earthshaking, his explanation does place management above

the detailed, technical work — looking down. This, he claims, makes management the most arduous in software development task — and consequently, the primary source failure. [BLUM92] In his book, *The Five Pillars of TQM*, General Bill Creech (USAF retired) makes the same observation:

“You must do more than talk about it; you must change the organization ‘conceptually’ and ‘structurally’ to bring leadership alive at all levels. Principles flow from the top down; decisions flow from the bottom up.” [CREECH94]

In other words, management fails because decision making is coming from the wrong direction. We are trying to make decisions from the top down, when they need to come from the bottom up. To control it, *managers need a better understanding of the software product and the engineering process*. Software-intensive acquisitions fail due to the following (or a combination them):

- Technology-driven solutions,
- Unstable requirements,
- Software’s inherent complexity,
- Thinking automated technology will make up for poor engineering practice,
- Poor estimation of size, schedule, and cost,
- Inadequate software staffing, and
- The domino effect resulting from any combination of the above.

There are no quick, easy solutions to these major, oft-repeated problems. *If there were, these Guidelines would not be so thick!* The battle damage assessments in the previous section illustrate the severity of our problems. The interrelated and multifaceted practices you need to avoid and best practices you need employ to correct these problems are discussed throughout these Guidelines.

The success or failure of a major software-intensive acquisition program depends on a highly complex combination of factors, not all of which are under the control of the acquisition manager. As illustrated on Figure 2-2, these factors can influence the management of a program at the organizational, acquisition, and program levels. Critical failure/success factors are the crucial risk elements of a software acquisition program. They may be managerial, financial, technical, human, or political in nature. Program success or failure occurs as a result of both subtle and obvious interactions among all factors. When one or more factors are in a less than optimal state, they increase the likelihood that a program will fail, or worst case — *become a disaster*. [FLOWERS96]

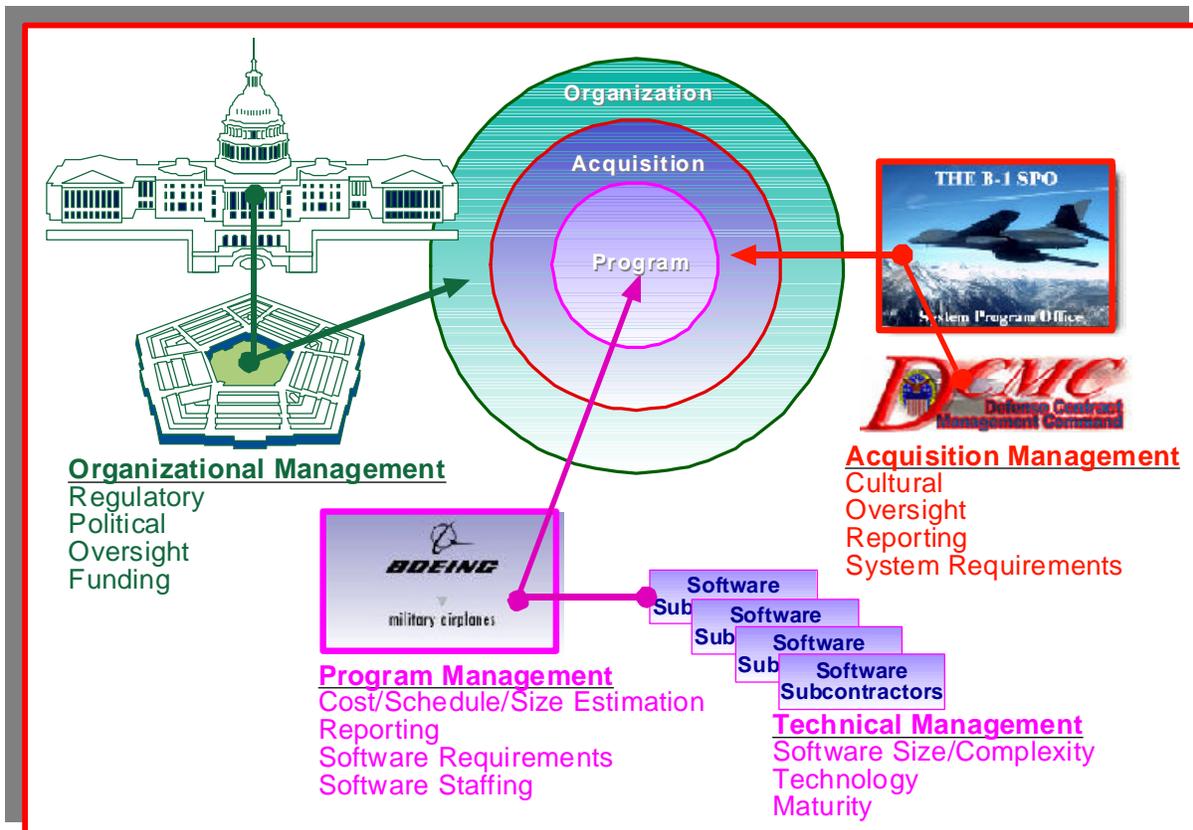


Figure 2-2. Examples of Interrelated Factors Affecting Software Acquisition Success

2.3.2 Technology-Driven Solutions

“[The] most critical aspect of profound military innovation is not technology, but understanding what we can do with it.” — SECDEF William S. Cohen [COHEN97]

To maintain our competitive edge and military superiority, software-intensive Defense systems often include performance requirements and design features demanding the acquisition of unprecedented technologies. *Joint Vision 2010* places DoD’s strategic vision in the lap of software technology. However, technology-driven solutions often result in acquisition disaster.

No one — not even the software developer — fully understands the implications of applying the unprecedented to a particular problem set. Thus, technology-based acquisitions are often a leap of blind faith, rather than pragmatic solutions. Flowers warns that the pure (black and white) world of software is often overturned by a costly realization of how messy the real world [battlefield] is.

DoD is not the only buyer with a predilection for technology-driven solutions. In 1995, KPMG reissued a 1989 survey to 250 large software development organizations in Great Britain, which focused on why software acquisitions fail. KPMG defined a software program failure as “a project that has failed significantly to achieve its objectives and/or has exceeded its budget by at least 30%,” [NOTE - Yourdon’s definition above was a 50% budget overrun.]

Because only half of the companies surveyed responded in 1995, KPMG surmised that industry had become reluctant to discuss software program failures. In 1989, only 7% of respondents thought technology was the cause of failure, whereas, 45% thought technology was to blame in 1995. KPMG concluded, “*Technology is developing faster than the skills of the developers.*” [COLE95]

With the complexity and array of software-intensive solutions, an educated, knowledgeable user is essential to acquisition success. When it comes to technology-based solutions, the warfighter must be encouraged to:

- Actively participate in defining the problem, and
- Assist in developing an appropriate solution.

2.3.3 Unstable Requirements

Essentially all software reports and studies concur that Requirements instability is a serious software failure factor. It is reasonable to assume requirements are going to change as user missions evolve in response to operational, threat, and technological changes. Thus, the first and most important factor to consider when managing unstable requirements is designing a software architecture with change-tolerant flexibility. It is also important not to accommodate subtle near-term requirements that can compromise the overall architectural design and limit future change. The second factor to consider is controlling the process and rate at which inevitable requirement changes are incorporated. If *ad hoc*, sporadic, or frequent modifications to requirements or their interpretation are inflicted on developers, *creeping changes in cost and schedule* are a given. In addition, what sometimes appear to be minor changes have dramatic *side effects* elsewhere in the software system. Full (technical and effort) evaluation of the consequences of each change must be included in the management process (in addition to configuration management control). Ad hoc incorporation of changed requirements will invalidate original estimates of cost and schedule, and impact product quality.

The fact that software is soft and changeable is also why software maintenance costs are more exorbitant than for other disciplines. According to Glass, we fail to realize that although software is soft, “*it is not so soft that change is free. Far from it, in fact. Change is the biggest money-maker in the software world!*” [GLASS91] We also do not realize that unstable requirements are negative characteristic of the software beast. We fail to freeze requirements at the outset of the program — when the Software Requirements Specification (SRS) is approved. If requirements keep evolving as the software is built (and especially if there is concurrent hardware development), it is next to impossible to develop a successful product. Software engineers find themselves shooting at a moving target and throwing away design and code faster than they can crank it out.

2.3.3.1 Inadequately Stated Requirements

One source of instability is “*inadequately stated*” requirements. Indefinite and undefined software requirements also lead to creeping cost and schedule changes which can continue even after the system enters development. The most important, yet difficult, software development task, requirements definition and analysis, plagues software managers in all industry sectors. Ill-defined requirements lead to poor specifications, which impact cost, schedule, and ultimately quality and

user satisfaction. Requirements creep has the greatest impact on our ability to produce accurate estimates [discussed below]. Figure 2-3 illustrates how the ability to predict software cost increases as requirements become progressively better defined. [BOEHM81] The two lines show the high and low estimate ranges. In the beginning of a project, during the feasibility study phase, cost estimates range anywhere between one fourth (.25) the real cost and four times the real cost. Estimates get better as the project progresses until the cost is known with exactness at the end.

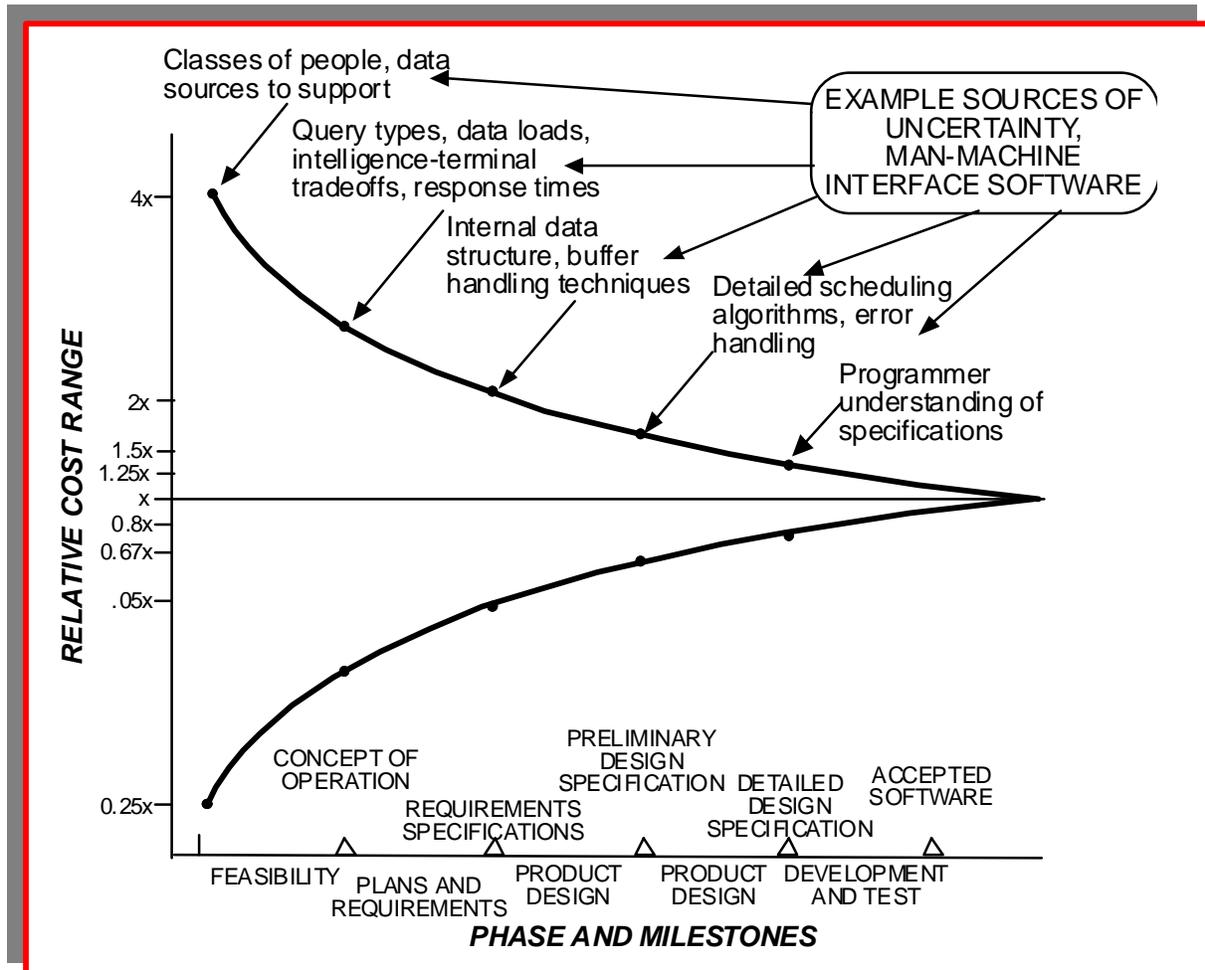


Figure 2-3. Software Cost Estimation Accuracy versus Phase [BOEHM81]

Requirements creep, something an acquisition group *can* manage and control, was a serious C-17 Globemaster development risk.

Although developed using mostly non-developmental items (NDI) and commercial-off-the-shelf (COTS) software, creeping performance requirements significantly increased technical risk. Some of the changes in software requirements were driven by manufacturing and weight problems that were resolved by the incorporation of new, lighter weight subsystems that were software intensive. For example, when the C-17 development program began in 1985, the Government planned the development of 4 subsystems with about 164,000 lines-of-code. By 1990, this number had increased to 56 subsystems and about 1,356,000 lines-of-code, including approximately 643,000 newly developed lines-of-code, as illustrated on Figures 2-4 and 2-5. [HINTON98]

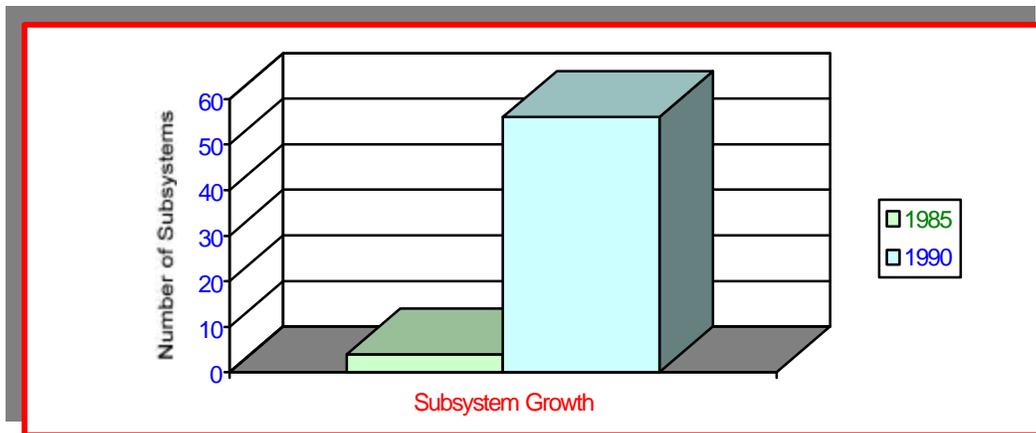


Figure 2-4. C-17 Software Subsystem Growth 1989-1990

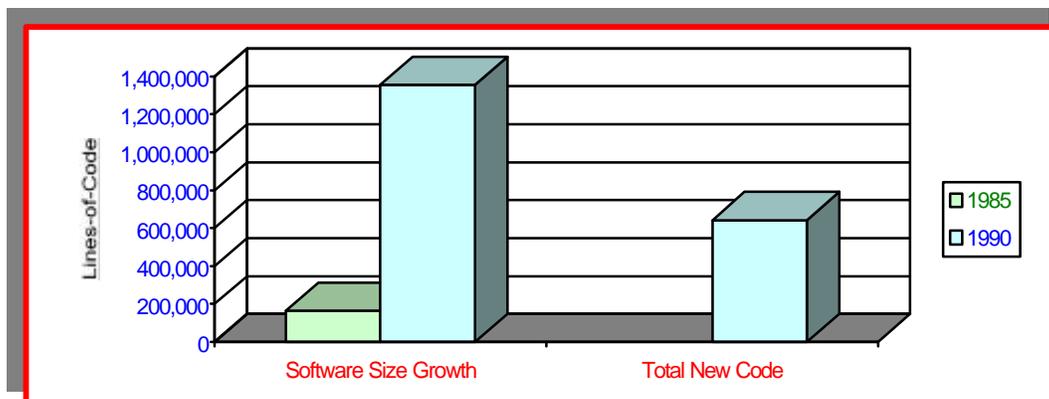


Figure 2-5. C-17 Software Size Growth 1989-1990

2.3.3.2 Inadequate User Involvement

Paul Paulson, president of Doyle, Dane, and Bernbach, a large New York brokerage firm, was quoted in the *New York Times* as saying,

“You can learn a lot from the client. Some 70% doesn’t matter, but that 30% will kill you.”
[PAULSON79]

Misinterpretation of user requirements is a major, if not the greatest, contributor to software failure. Not understanding your client (the military user) while managing software development is one sure way to make your program *crash-and-burn*. Misunderstood requirements are also the source of costly support problems.

User involvement is critical throughout requirements analysis and design, where feedback is essential to determine whether perceived user needs have been correctly translated into software functionality. The 1992 final report of the Software Process Action Team found that the primary reason major software-intensive programs fail was the *inability to translate user needs into viable software requirements*. The report states:

“The procurement process often results in government acquisitions that fail to meet user needs. The problem is exacerbated during system development when requirements decisions are made without adequate user input and without full understanding of the overall impact on costs, schedules, performance, and other critical factors. Current government and industry practices have led to requirements specifications that contain design information, inappropriate levels of detail, inadequate requirements, and poor traceability.” [PAT92]

Lessons-learned from the Air Force’s Nuclear Mission Planning and Production System (NMPPS) warn that users often do not start out with a *clean slate* when explaining their operational, readiness, and logistics requirements. Additionally, they may view the statement of their requirements as one more document being coordinated within headquarters, which can be changed with minimal impact. Personnel with operational experience can also contribute to the problem because they, too, assume they generally know the requirements and need to ask users fewer questions. [KEENE91]

Another example of the user involvement issue occurs when the program office designates responsibility for defining system requirements to someone other than the user. These *user representatives*, often called functional analysts, use a systems analysis approach to functional system design. While some functional analysts have extensive backgrounds in the target system, others rely on a limited understanding of user requirements. In both cases, understanding user requirements quickly diminishes without frequent exposure to the target system’s operational environment and its users.

Once developed, functional specifications are passed on to programmers who must interpret them and write the code. Large programs can have 50 or more programmers receiving functional guidance using this method. Considering that initial guidance is likely to be partially flawed at best, a second translation compounds the situation. During system testing, efforts are expended in determining whether a software error was introduced during functional design or a defect during coding. The likely result of this design-to-product process is the most costly software failure — *software redesign and rework*. [HENDERSON95]

NOTE - Throughout these Guidelines, the distinction is made between the terms “error” and “defect.” They are defined as:

Error: A mistake inserted during design.

Defect: A mistake inserted during coding.

2.3.3.3 Poor Communications

The main reason errors occur during requirements definition and analysis is poor communications. During the requirements phase, the user tries to articulate a concept of expected system function and performance into concrete detail. The software engineer attempts to translate user definitions into models of information, control flow, operational behavior, and data content. The chances for misinterpretation, misinformation, and ambiguity are numerous. General John W. Vessey, when serving as Chairman of the Joint Chiefs of Staff, explained that

“More has been screwed up on the battlefield and misunderstood at the Pentagon because of the lack of understanding of the English language than any other single factor.” [VESSEY84]

Lack of understanding of what software is, how it performs, and difficulty in conveying what it is to do, are compounded by inherent shortcomings of the English language. The dilemmas confronting software engineers are expressed in the user's statement, "I know you believe you understood what you think I said, but I'm not sure you realize that what you heard is not what I meant." [PRESSMAN92] Under these conditions, designers fail to translate conceptual user needs into functional software requirements. Software that does not fit user needs is destined for the trash heap! Brooks explains,

"The hardest single part of building a software system is deciding precisely what to build. No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later." [BROOKS87]

Data collected at Rome Laboratory indicate that over 50% of all software errors are "requirements errors." Requirements errors are more expensive to correct the further they percolate throughout the life cycle. It is often 50 times more expensive to correct a defect during systems integration than during requirements analysis. [DiNITTO92] Figure 2-6 illustrates how the cost to correct requirements errors increases during subsequent phases of development.

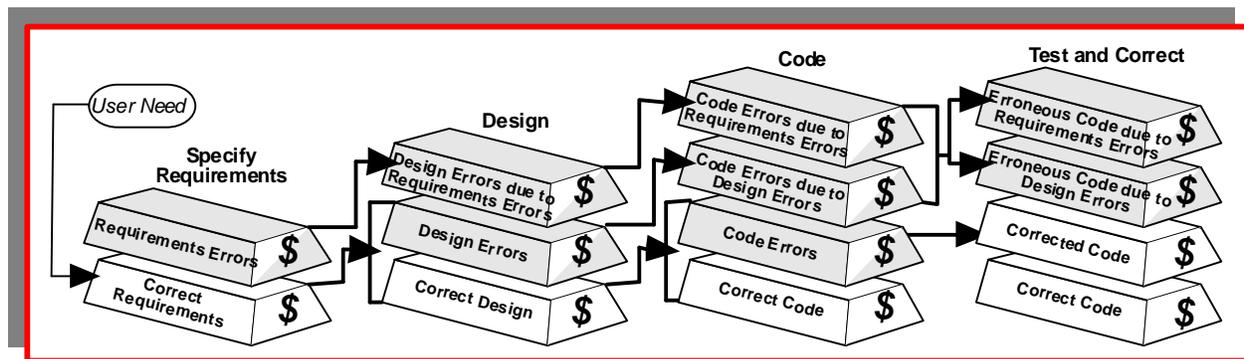


Figure 2-6. Error Propagation Cost

NOTE - See Chapter 11, *Understanding Software Development*, for an in-depth discussion on system and software requirements.

2.3.4 Software Complexity

The complexity issue arises when confronted with the enormous tasks we often want our software to perform. With the Strategic Defense Initiative (SDI), for example, we undertake giant, unique developments that require years of effort and hundreds of people to produce. Being unprecedented, they cannot be built using previous knowledge and cannot be tested under actual operational conditions or in the environment in which they will be used. Nailing down how the software should perform under these circumstances is often problematic.

The irony with software is that, while it has automated just about every labor-intensive activity known to man, software is still handmade. In addition to being handmade and hard to build, it is inherently prone to human error. As Parnas and Brooks explained, software is risky because it is *hard* to build. The complexity of hardware pales in comparison with that of software. For any given hardware problem, there is a high percent of component reuse and a finite number of

solutions. With software — even with optimized solutions — there are a near-infinite number of possible correct solutions. Theoretically, any set of problems from any other discipline can be solved within the software domain. [GLASS92]

2.3.4.1 Size and Complexity

“The business of creating new computer software — the programs that make computers work — is one of the most complex, painstaking, even exasperating jobs around. It is as if someone is writing War and Peace in code, puts one letter out of place and turns the whole book into gibberish.”
—Robert N. Britcher [BRITCHER98]

Size and complexity go hand in hand. The bigger the application, the more complex it becomes. Complexity plagues us because we often fail to take a disciplined approach to design and thereby create more complexity than needed. Although sometimes necessary to match problem complexity, software size and complexity must be kept to the minimum. Ed Yourdon places odds on successful program completion by size. By software development program size, the odds of success are as follows.

Program Size	Number of People	Length of Program	Odds of Success
Small scale	<10	3-6 months	High
Medium-sized	20 – 30	1-2 years	Slight
Large scale	100 – 300	3-5 years	Bleak
Mind boggling	1,000 – 2,000	7-10 years	Doomed

Table 2-10. Odds of Successful Completion by Software Team Size [YOURDON97]

NOTE - These Guidelines are applicable to all software acquisitions; however, the main focus is on Yourdon’s large-scale to mind boggling size programs.

David McLure, GAO’s assistant director, confirms that, *“The government’s track record on large systems is bleak.”* [McLURE98] In 1987, Tom DeMarco and Timothy Lister, reported that the cause is sociology factors! [DeMARCO87]

In addition, highly complex solutions are destined to be high-cost *maintenance nightmares!* The bigger, the more complex the software — the more difficult it is to understand — the greater the chance that defects will propagate throughout the code. The cost of making changes and correcting defects often soars beyond acceptable levels, resulting in programs abandoned after exorbitant expenditures of unrecoupable resources. According to James Johnson, chairman of The Standish Group,

“Software projects are far more likely to be successful if they’re highly focused and built upon well-understood technology. If you can reduce the scope of the project, your chances of success are far greater.” [JOHNSON98]

Brooks explains that the best designs are those that *“produce structures that are faster, smaller, simpler, cleaner, and produced with less effort.”* [BROOKS87]

Size and complexity not only create technical problems, they create serious team and management problems. Fernando Corbato summarizes the size/complexity problem with the following.

“The most obvious complexity problems arise from scale. In particular, the larger personnel required, the more levels of management there will be...The difficulty is that with more layers of management, the top-most layers become out of touch with the relevant bottom issues and the likelihood of random serendipitous communication decreases. Another problem of organizations is that subordinates hate to report bad news, sometimes for fear of ‘being shot as the messenger’ and at other times because they may have a different set of goals than the upper management. And finally, large projects encourage specialization so that few team members understand all of the project. Misunderstandings and miscommunication begin, and soon a significant part of the project resources are spent fighting internal confusion. And, of course, mistakes occur.”
[CORBATO92]

2.3.4.1.1 Automated Software Development

Software technology has been the greatest instrument for improving man’s efficiency since the Industrial Revolution. When properly used, it provides remarkable competitive advantage. Paradoxically, while software significantly increases the efficiency of its users, the way software is produced is quite inefficient. Not only is software handcrafted — it is produced by *manual labor*! Where automation has achieved the most significant increases in human productivity, little progress has been made in automating the software process. According to Capers Jones,

“The problem is that software has the highest manual labor content of almost any manufactured item in the second half of the 20th Century.” [JONES90]

Increasing software productivity and quality is the greatest challenge to the software community. We must learn to produce software cheaper, better, and faster. In our quest for more efficient methods, we sometimes fail to realize there are no easy solutions. We often focus too much on software’s potential, real or imagined, and ignore its limitations. As Brooks explains,

“...as we look to the horizon of a decade hence, we see no Silver Bullet. There is no single development, in either technology or in management technique, that by itself promises even one order-of-magnitude improvement in productivity, in reliability, in simplicity.” [BROOKS87]

Why do unproven methods and technologies cause program failures? Glass tells us *“the search for magic solutions diverts us from the more important search for mundane ones.”* We neglect proven reliable solutions and invest in the hope that a *pie-in-the-sky* magic one will emerge. [GLASS92] They make us focus all our attention on one method or technology that promises vast improvements, rather than implementing proven ones in parallel. When building large, complex software-intensive systems, it takes more than just one tool or technology advance for significant process improvement. Multifaceted approaches, including tools, methods, techniques, and processes used in parallel are the proven way to quantifiable progress. [JONES94]

Unproven methods and tools are also the reason why software technology transfer has been so slow. Rarely have these methods been successfully transferred from the laboratory to the production line. The reason many wash out is few can scale up to the demands of large, software-intensive developments. While new technologies can increase productivity, *we fail in acquiring and managing them.* We tend to jump on the hype bandwagon, select, and acquire them without

detailed knowledge of their impact on the development process. Once a financial commitment has been made, we find these tools do not blend in with established processes. We do not anticipate the extra time and resources required to train software personnel on the new processes required by the tools. In addition, we may have to reengineer our old process to fit the one imposed by the new technology. Many a program has failed because tool selection is not based a needs-driven process and a pre-acquisition determination that they will be beneficial to the people who must use them. [See Chapter 11, *Understanding Software Development*, for a more detailed discussion on software tools.]

2.3.5 Poor Estimates

Management, like all other software activities, is a problem-solving exercise. It involves deciding what must be accomplished, how to do it, monitoring what is being performed, and evaluating what has occurred. The “*what*” is expressed in the Software Development Plan (SDP) [see Chapter 11, *Understanding Software Development*] and the “*how*” in the allocation of resources (e.g., schedule and budget). Too often, we stop after these first two steps. We do not remember that software development is dynamic and our original plans and estimates must be constantly updated. We need more time than projected. New requirements are added. Key personnel are sent to other programs. We fail to monitor activities and adjust our plans and resources accordingly. [BLUM92] For example, when change requests are submitted, we fail to make a solid estimate of their impact on our cost and schedule predictions and to change those figures to accommodate the new requirements. We fail to tell our customers (the warfighter or the Government, if you are a contractor) that if they want a change badly enough, *they will have to pay for it*. It is easy to understand this problem, but few managers act on it. [GLASS92] We are caught up in trying to please and wind up playing a fatal game of catch-up.

“[T]he most common problem in building software systems is not the construction of them itself, but rather the estimation of the costs of that construction. Why is there such a problem of estimation? Because the software field has not made a conscientious effort to develop histories of past project costs. Because the construction of software is an extremely complex task — some say it is the most complex task ever undertaken by human beings. Because the lack of history and the amount of complexity, a barrier was produced that no amount of mathematical techniques and no amount of individual expertise has been able to overcome. It is all too common for a software project to fail to meet its cost and schedule targets, because the targets themselves were simply (and grossly!) wrong.” — Robert L. Glass [GLASS98]

The fundamental reason software-intensive developments overrun cost and schedule, resulting in quality and performance shortfalls, is our *inability to estimate or to establish realistic program baselines*. No matter how smooth the development process, how efficient the tools, or how smart the designers, our predictions of cost and schedule are frequently out of sync with what actually occurs in the production of a software product, or politics dictate the establishment of a baseline with insufficient funds, schedule, or both. We often forget that software development involves much more than simply writing code. For example, we are still learning that software inspections and testing take longer than anticipated and that maintenance consumes between 60% and 80% of the software dollar. We also do not account for the cost of *scrap* and *rework* involved when a developer has an ad hoc, chaotic development process. Boehm claims the cost for rework to be about 44% of every dollar spent, as illustrated in Figure 2-7. [BOEHM81]

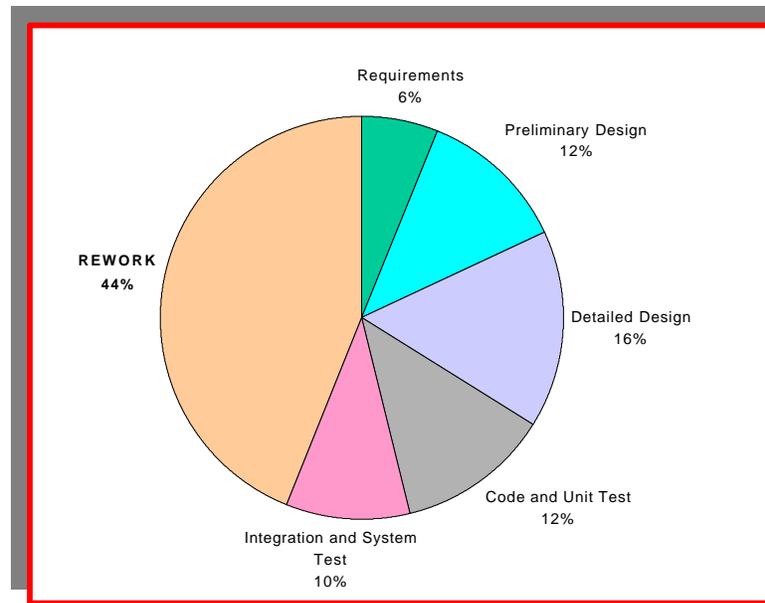


Figure 2-7. Defect Rework Compared to Other Development Costs [BOEHM81]

2.3.5.1 Size/Complexity Estimates

Predicting the size and complexity of the software to be built is at the heart of our estimation difficulties. As mentioned above, software is intangible — we cannot weigh it, box it, put our arms around it, or paint a picture of it. Thus, our attempts to project how big it will be, how hard it will be to create, or how long it will take are problematic. When software is preceded (i.e., a similar system has been built) size and complexity projections (and thus, cost and schedule) are usually more accurate. In unprecedented systems, however, our inability to estimate the intangible is acute.

2.3.5.2 Cost/Schedule Estimates

A 1993 RAND Corporation study concluded that over the years, there has been no improvement in controlling cost growth on the average weapon system. [RAND93] These overruns persist despite the implementation of Defense initiatives to mitigate cost risk and growth, including the significant risk management guidelines DoD instituted in 1985 to improve the transition from development to production. There is often a substantial discrepancy between what was originally estimated to develop and produce a system (per unit) and what the system (per unit) costs once in production. This happens because all factors are not considered, history for a similar project is unreliable or not available, or too much optimism is injected into the process to make the costs more palatable.

2.3.5.3 Optimistic Estimates

“I cannot imagine any condition which could cause this ship to flounder. I cannot conceive of any vital disaster happening to this vessel.” — E.J. Smith, Captain of the Titanic, 1912

“Titanic Effect: The severity with which a system fails is directly proportional to the intensity of the designer’s belief that it cannot.” — ACM SIGSOFT, 1986

In DoD, we are subject to spending and budget scrutiny from the Congress, the press, the public, and upper management. Under pressure, contractors and military managers often make overly optimistic estimates about how much the software will cost and how long it will take to produce. It is human to discard pessimistic cost, schedule, and size estimates and base projections on the *best of all possible worlds*. We fail to manage risk and build a *management reserve* or *worst-case scenario* into cost/schedules for fear our programs will not be approved or funded if we submit figures that are more realistic. Within the framework of the annual Defense budget, problems or indications that program estimates are decaying do not help sustain it in subsequent years, and thus their admission is discouraged.

An optimistic program cost estimate makes it easier to launch a software-intensive development and sustain annual approval; admission that costs are likely to be higher could invite failure. [HINTON98] This acquisition culture compounds the likelihood for shortcuts in the development of a system that was improperly funded and scheduled. Also, in these cases the problem is often not the actual cost, schedule, and size estimates (which in many cases may be reasonable) — it is the failure to use these estimates to establish reasonable, attainable program baselines.

“I cannot give you the formula for success, but I can give you the formula for failure—which is: Try to please everybody.” — Herbert B. Swope [SWOPE50]

2.3.6 Inadequate Software Staffing

“We have suddenly discovered that our great national resource is not our mines, mills and factories, but people. People with ideas, energy, ambition. Creative people. People who are willing to take a chance. People who can be motivated to excel.” — Harold S. Longman [LONGMAN92]

Success or failure in software acquisition depends on the skills, experience, and ingenuity of the people who build, implement, and maintain it. The winners and the losers in the global security environment will be those nations with the best people to invent, engineer, and put that software technology to work for the warfighter. The old axiom rings true, *“an idea can turn to dust or magic depending on the talent that rubs against it.”* [BERNBACH82] In 1979, the GAO made some stirring comments about the state of software development, which unfortunately, still ring true. It stated that,

“[s]everal factors contributed to the [software development] situation. First, the invisible nature of both the work process and its product made software projects very difficult to manage and predict. Second, the explosive growth of the use of computers created demand for new programmers, most of whom were self-taught on the job; and frequently, low productivity and poor quality resulted. Third, there was little idea then of how to train programmers properly. Fourth, a tradition grew that programmers were secretive craftsmen, whose products, during development, were their own property.” [GAO79]

2.3.6.1 Software Labor Shortage

The Software Program Manager’s Network (SPMN), a tri-service (Army, Navy, and Air Force) organization (with Navy as lead), was formed in response to the 1994 Software Best Practices Initiative [see Chapter 4, *DoD Software Acquisition Environment*]. The SPMN mission is to

address software development and maintenance problems on major software-intensive programs DoD-wide. One particular problem with which the SPMN is grappling is the shortage of skilled software workers on Defense acquisition programs. Congress has directed the SPMN to assist “*in the development of industry and academia pilot projects to attract, train, and retain skilled software personnel for software-intensive projects within the Navy and DoD.*”

In response to this directive, the SPMN formed the **National Software Alliance (NSA)**. In the NSA report, *Software Workers for the New Millennium: Global Competitiveness Hangs in the Balance*, the software worker shortage is described as being two-fold. [JOHNSON98] First, there are not enough software professionals to fill escalating labor demands. Second, there is a shortage the highly specialized software skills and experience needed by employers. The NSA summarizes the demand for software as follows:

The three software occupations classified by the U.S. Department of Labor are: computer programmers, computer systems engineers, and computer scientists (database administrators, computer support specialists, and all other computer scientists.)

- 72% of all U.S. software workers are employed by non-software related industries; only 28% of software workers are employed by software goods and service industries.
- Between 1996 and 2006, total U.S. employment is projected to increase by 18.6 million jobs, a 14% growth rate. During the same period, total software worker employment is projected to increase by 75%, from 1,502,000 to 2,634,000 software workers.
- Between 1996 and 2006, U.S. employment is projected grow at an annual rate of 1.3% for all industries. During the same period, computer and data processing services employment is projected to grow at an annual rate of 9.3%, seven times faster than the national average.
- Between 1996 and 2006, the three software occupations are projected to be the fastest growing occupations in the U.S. economy.
- Computer engineers will grow 109%, from 216,000 to 451,000 workers.
- Systems analysts will grow 103%, from 506,000 to 1,025,000 workers.
- Database administrators, computer support specialists, other computer scientists will grow 118%, from 212,000 to 461,000 workers.
- Between 1996 and 2006, in addition to the newly created jobs, 245,000 replacement workers will be needed to fill the jobs of software workers leaving the field.
- Between 1996 and 2006, the computer and data processing services industry (the leading employer of software workers) is projected to be the fastest growing U.S. industry, increasing 108%, from 1,208,000 to 2,509,000 workers. (The next fastest growing industry, health services, will increase 68% over the ten-year period.)

Throughout this decade, the demand for software professionals has grown at an unprecedented rate. In 1996, virtually every industry in the U.S. economy employed software workers — about 1.5 million in all. [SILVESTRI97] These industries are competing fiercely for a finite labor pool, driving up salaries, bonuses, and stock options, in an often ineffectual attempt to attract and retain employees. A number of factors contribute to the increasing demand, such as growth of the Internet, electronic commerce, the software export market, high-tech industries, and the Year 2000 fix. Market expansion in areas such as microelectronics, biotechnology, machine tools and robotics, aerospace, and other software-intensive industries have also intensified pressure on the pool of software professionals. As the demand for software increases, the demand for qualified

software workers will escalate at an even faster rate. The NSA summarizes impacts of the software labor shortage as follows.

- 59% of high-tech companies report they are understaffed;
- 62% of high-tech companies plan to increase staffing.
- 61% of high-tech companies report that a lack of properly skilled applicants is their greatest staffing challenge.
- 66% of high-tech companies identify staffing problems as their primary barrier to growth.
- The average annual turnover rate for software professionals is about 14.5%.
- Many software “hot jobs” have annual salary increases of 15% to 20%.
- In 1996, the number of unfilled information technology jobs was estimated to be between 190,000 and 450,000 open positions nationwide. [JOHNSON98]

2.3.6.2 Defense Software Jobs

DoD, the world’s single greatest consumer of software, has been the hardest hit of all federal agencies by the software labor shortage. The impact on DoD in-house software maintenance and development has been particularly severe, where government salaries are not competitive with those in the private sector. A July 1998 report by Peter Jennings on *ABC World News Tonight* highlighted the situation in the Air Force.

“And more bad news for the Air Force, which prides itself on high technology, can be found buried deep in the computer rooms that are the nerve centers for the force. Chief Master Sergeant Steve Lovin says he can no longer hang on to any computer specialist. A typical first-term airman who’s making \$20,000 a year can get anywhere from \$60,000 to \$80,000 a year starting working for industry at this point.” [JENNINGS98]

Jennings interviewed a typical computer specialist, Regina Nienaber, who was leaving the Air Force after her first three-year commitment. He asked if many people at her experience level were reenlisting. She replied that in the last 24 months, only one 1 out of 12 of her co-workers was staying in the service. According to Jennings,

“[t]he other services have similar problems. The Army and the Navy are losing highly trained technicians and computer specialists to the private sector at about the same rate as the Air Force. The Defense Department tells us that military salaries are, on average, 14% below comparable jobs in the private sector. And the Pentagon estimates it would cost \$30 billion to make them equivalent. The Army will offer a \$6,000 bonus to a computer specialist who will reenlist. But it doesn’t compare to what they can get on the outside.” [JENNINGS98]

2.3.6.3 Labor Shortage Impacts

As explained throughout this chapter, software acquisition failure/success factors are interrelated. Less than optimum conditions in one factor bleed over and disrupt the equilibrium of others. Brooks explains,

“Not only technical problems but management problems as well come from the complexity [of software]...It makes it hard to find and control loose ends. It creates the tremendous learning and understanding burden that makes personnel turnover a disaster.” [BROOKS87]

Staffing problems result in program delays, cost overruns, and poor software quality. In fact, DoD felt the impact of the software labor shortage as far back as 1990. An article in *Government Executive* stated,

“Rep. John Murtha (D-PA) feels...software is the technology that threatens the nation’s military dominance...Between the vast complexity of such programs [F-22 Advanced Tactical Fighter] and a severe shortage of software engineers to write them, many weapons projects are falling seriously behind schedule...The software crisis is in large part a personnel crisis. And the unnamed coconspirators in it are the nation’s universities. Universities don’t seem to understand what a software engineer is...The bottom line is that software has generated hell on earth for project managers, leading to delays, cancellations, lost jobs, and huge cost overruns. And all those interviewed agree that the best they can hope for in the next 5 to 10 years is that the temperature of their hellish environment will lower from ‘burn’ to ‘roast.’” — Alton Marsh [MARSH90]

The hellish software personnel problems that plagued DoD in 1990 persist. In fact, temperatures in the software arena are on the rise. DoD, once the hotbed for high-tech innovation, is being outpaced by the commercial sector where success is defined by the effective adaptation of the most powerful, advanced software-intensive technologies. With limited defense dollars, DoD is losing a recruitment war with the commercial sector for the best and brightest software professionals upon which it critically depends.

DoD contractors are handicapped by the same budget constraints as their defense customers, and hobbled by federally established labor rates intolerant of extraordinary salary surges in any particular labor segment. Defense contractors are also restricted from outsourcing DoD software development to foreign workers by the Federal Acquisition Regulation (FAR). Their ability to compete with escalating commercial sector salaries is also limited by the corporate need to make a profit often bound by fixed-price and/or fixed-fee contracts. The problem is exacerbated in locations where DoD and Defense contractors compete with the private sector for highly-skilled software workers in already tight labor markets, such as California and metropolitan Washington, D.C. [PIETRUCHA97]

2.3.6.4 DoD Hardest Hit by Shortage

The demand for software professionals is the most acute in non-software industry and government sectors. These jobs, where the majority of software workers are employed, are less desirable, pay less, and are often more difficult to fill. As illustrated on Figure 2-8, DoD software positions are among the least desirable, the hardest to fill, and represent the greatest demand. [JOHNSON98] Also shown is the segmenting of the national demand for software workers into three tiers by Avron Barr and Shirley Tessler, from the Stanford University Computer Industry Project. [BARR98]

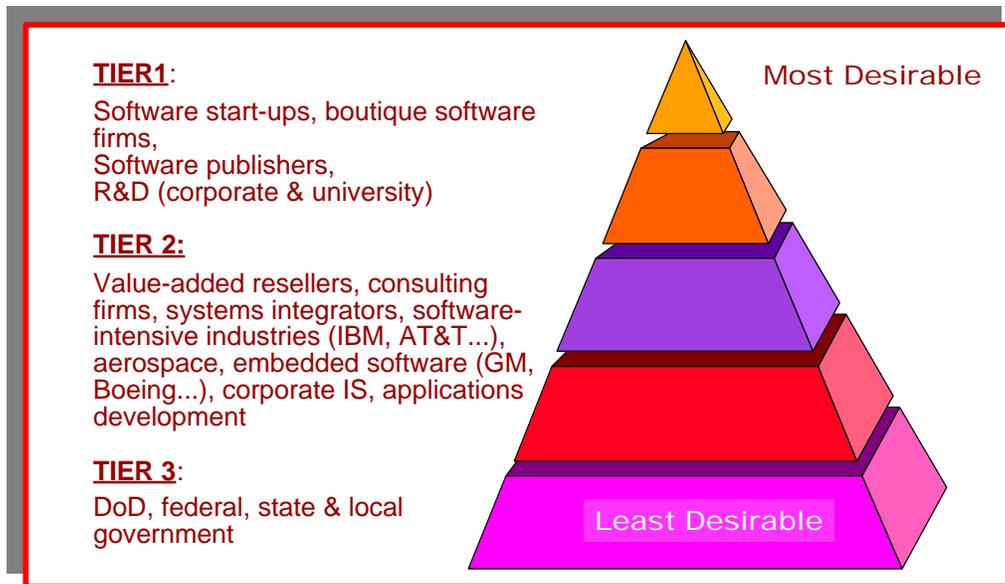


Figure 2-8. Software Worker Employment Tiers. Size of Tier Represents Demand. [BARR98]

- **Tier 1** - The top tier, representing less than 10% of the workforce, is comprised of the venture capital-funded software startups, “boutique” software service companies, and software publishing organizations. Big salaries, signing bonuses, equity sharing, and challenging work environments draw the best and brightest from other industry segments and the colleges. This tier, Barr and Tessler claim, is growing at an annual rate of 15% and has been the least impacted by the software labor shortage.

Jeffrey Bier, Vice President of Lotus Development Corporation, described Tier 1 software workers in a speech, “Managing Creatives,” at Industry Week’s annual Managing for Innovation Conference.

“Creatives are intense. They’re always thinking about work. For them, there’s no such thing as ‘Miller Time.’ They think in their sleep...They are happy to come to work every day and solve puzzles. As one of my people says, ‘You come in every day and you’re given a set of games to play. Fifteen puzzles. Things don’t fit and you’ve got to make them fit.’ To the creative person, that’s heaven!...In general, they work for three things. First, the ‘fun’ of creation itself. Second, ‘admiration’ — especially from their peers. Third, the excitement and ‘glory’ of taking part in a successful creation.” [BIER95]

- **Tier 2** - This second tier includes computer and high-tech equipment manufacturers, telecommunications, financial services, and other software-intensive industries. While market demand has increased software salaries, this tier is having difficulty recruiting and retaining the best and brightest that are lured away by savvy Tier 1 recruiters. Because jobs in these firms are not as glamorous, nor are software workers as revered as they are in Tier 1, the majority of Tier 2 jobs are filled by whomever companies can find to accept the salaries offered.

- **Tier 3** - This tier is at the bottom of the software worker supply chain. One of the largest employment segments, it includes most government and manufacturing organizations. In Tier 3 the strategic importance of software and a strong software workforce has not reached executive-level awareness. This tier lacks a competitive response to the tight labor market. Tier 3 has been the hardest hit by the labor shortage because workers (once they have gained skills and experience) readily move from these positions to Tier 2, where salaries and working conditions are more desirable. [BARR98]

2.3.7 The Domino Effect

Programs tend to get in trouble in small, but progressively compounding increments. Schedule changes, due to unrealistic estimates of development time required, often start as unnoticeable changes in plans that go undetected by most managers. Schedule slips, starting small, have the potential of becoming major problems because even small slips impact the delivery of other related elements and almost always affect cost. Late software (on the system's critical path) has a *domino effect* on other system components, which have to slip their schedules while waiting for the delinquent software. By failing to recognize and deal with this problem with expeditious corrective action, the situation can quickly deteriorate into a *software disaster!*

When the product is late, we apply management pressure to reduce the slack between our projected delivery date and the illusive real one. This aggravates into a *Catch-22* situation. With inadequate resource and schedule estimates, the time required to *build quality in* may be insufficient. In addition, to meet schedule and keep down cost, the easiest thing with which to economize is testing. [GLASS92] Before we realize it, a late, over-cost program evolves into an unreliable one. From the developer's point of view, when a cost/schedule disaster is discovered, they often try to protect their contract with alternative proposals attempting to deliver less for the same price. This leads to *down-scoping* and eliminating or trading off requirements, to stay within initial projections. [MARCINIAK90] This is a very serious situation because it means resources have been expended, or often exhausted, and the user does not get the system for which was paid for. After years of schedule and cost overruns many programs have been canceled without the delivery of a single line-of-code.

Poor performance in one risk area translates into problems in other areas. For example, software quality directly relates to the quality of the management and engineering processes, because quality problems quickly become exorbitant cost problems. Once the software is delivered and in the user's hands, latent defects often need correction. Most defense systems have long operational lives, the software of which must be modified to adapt to new or changing requirements, and upgraded to new technologies. In general, our software is so unique, custom-crafted, and poorly documented, the only one who can figure it out is the original designer — who is often working on something else or otherwise unavailable. This translates into software that imposes heavy training loads and high labor costs.

2.4 References

- [ADAMS96] Adams, Scott, *The Dilbert Principle*, Harper Collins Publishers, New York City, New York, 1996.
- [BARR98] Avron, and Shirley Tessler, “How Will the Software Talent Shortage End?” *American Programmer*, January 1998.
- [BERNBACH82] Bernbach, William, as quoted in the *New York Times*, October 6, 1982.
- [BLUM92] Blum, Bruce I., *Software Engineering: A Holistic View*, Oxford University Press, New York, 1992.
- [BOEHM81] Boehm, Barry W., *Software Engineering Economics*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1981.
- [BRITCHER98] Britcher, Robert N., “Why (Some) Large Computer Projects Fail,” included in Robert Glass, *Software Runaways: Lessons Learned from Massive Software Project Failures*, Prentice Hall PTR, Upper Saddle River, New Jersey, 1998.
- [BROOKS87] Brooks, Fredrick P., Jr., “No Silver Bullet: Essence and Accidents of Software Engineering,” *Computer*, April 1987.
- [BUSWEEK85] “Forget the \$400 Hammers: Here’s Where the Big Money is Lost,” *Business Week*, July 8, 1985.
- [COHEN97] Cohen, SECDEF William S., Keynote Speech presented at the National Defense University Joint Operations Symposium QDR Conference, Fort McNair, Washington, D.C., 23 June 1997.
- [COLE95] Cole, Andy, “Runaway Projects — Causes and Effects,” *Software World*, Vol. 26, No. 3, UK, 1995; as quoted in Robert L. Glass, *Software Runaways: Lessons Learned from Massive Software Project Failures*, Prentice Hall, Upper Saddle River, New Jersey, 1998.
- [CONAHAN95] Conahan, Frank C., “Defense Programs and Spending: Need for Reforms,” Testimony Before the Committee on the Budget, House of Representatives, GAO/T-NSAID-95-149, April 27, 1995.
- [CONE98¹] Cone, Edward, “[Federal CIOs Look Past Failures](#),” *InformationWeek*, January 12, 1998.
- [CONE98²] Cone, Edward, “[Crash-Landing Ahead](#),” *InformationWeek*, January 12, 1998.
- [CORBATO92] Corbato, Fernando, “On Building Systems That Will Fail,” *Communications of the ACM* 34, No. 9, September 1992.
- [CREECH94] Creech, General Bill, *The Five Pillars of TQM: How to Make Total Quality Management Work for You*, Truman Talley Books, Button, New York, 1994.
- [DeMARCO87] DeMarco, Tom and Timothy Lister, *Peopleware: Productive Projects and Teams*, Dorset House Publishing Co., New York, 1987.
- [DiNITTO92] DiNitto, Samuel, A., Jr., “Rome Laboratory,” *CrossTalk*, Software Technology Support Center, Hill AFB, Utah, June/July 1992.
- [DSB87] “Report of the Defense Science Board Task Force on Military Software,” Office of the Under Secretary of Defense for Acquisition, September 1987.
- [DSB94] “Report of the Defense Science Board Task Force on Acquiring Defense Software Commercially,” Office of the Under Secretary of Defense for Acquisition & Technology, June 1994.
- [FLOWERS96] Flowers, Stephen, *Software Failure: Management Failure*, John Wiley & Sons, West Sussex, UK, 1996.
- [GAO79] *Contracting for Computer Software Development—Serious Problems Require Management Attention to Avoid Wasting Additional Millions*,” FGMSD-80-4, United States General Accounting Office, Washington, D.C., November 9, 1979.
- [GAO95] *Tactical Aircraft: Concurrency in Development and Production of the F-22 Aircraft Should Be Reduced*, GAO/NSIAD-95-59, United States General Accounting Office, Washington, D.C., April 1995.
- [GATES95] Gates, Bill, “The Importance of Making Mistakes,” *USAir Magazine*, July 1995.

- [GIBBS94] Gibbs, W. Wyatt, "Software's Chronic Crisis," *Scientific American*, September 1994.
- [GLASS91] Glass, Robert L., Software Conflict: Essays on the Art and Science of Software Engineering, Yourdon Press, Englewood Cliffs, New Jersey, 1991.
- [GLASS92] Glass, Robert L., Building Quality Software, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1992.
- [GLASS98] Glass, Robert L., Software Runaways: Lessons Learned from Massive Software Project Failures, Prentice Hall PTR, Upper Saddle River, New Jersey, 1998.
- [GREENE90] Greene, Col. Joseph, Jr., as quoted by Evelyn Richards, "Pentagon Finds High-Tech Projects Hard to Manage: The Army Still Awaits Computerized Battlefield," *The Washington Post*, December 11, 1990.
- [GREENWALT98] Greenwalt, Bill, as quoted by Edward Cone, "Federal CIOs Look Past Failures," *InformationWeek*, January 12, 1998.
- [HENDERSON95] Henderson, COL Jerry M., "Will Army Software Win the Information War?" *Army RD&A*, July-August 1995.
- [HINTON98] Hinton, Henry L., *Best Practices: Successful Application to Weapon Acquisitions Requires Changes in DOD's Environment*, GAO/NSIAD-98-56, National Security and International Affairs Division, United States General Accounting Office, Washington, D.C. 20548, March 24, 1998.
- [JAMES97] James, Geoffrey, "IT Fiascos...and How to Avoid Them," *Datamation*, November 1997.
- [JCS96] *Joint Vision 2010*, Chairman of the Joint Chiefs of Staff, Department of Defense, The Pentagon, Washington, D.C., 1996.
- [JENNINGS98] Jennings, Peter, "Military Readiness: Can the Air Force 'Fight and Win Any War'?" *World News Tonight with Peter Jennings*, July 16, 1998.
- [JOHNSON98] Johnson, Susan Tinch, and Jack A. Bobo, *Software Workers for the New Millenium: Global Competitiveness Hang in the Balance*, National Software Alliance, Arlington, Virginia, January 1998.
- [JONES90] Jones, Capers, as quoted by Evelyn Richards, "Society's Demands Push Software to Upper Limits: More Computer Crises Likely," *The Washington Post*, December 9, 1990.
- [JONES94] Jones, Capers, Assessment and Control of Software Risks, Yourdon Press, Englewood Cliffs, New Jersey, 1994.
- [JONES95] Jones, Capers, Patterns of Software System Failure and Success, International Thomson Computer Press, Boston, MA, December 1995.
- [KEENE91] Keene, Charles A., White Paper "Lessons-Learned: Nuclear Mission Planning and Production System," Air Force Strategic Communications Computer Center (SAC), Offutt AFB, Nebraska, January 17, 1991.
- [LONGMAN92] Longman, Harold S., as quoted by Jerome B. Landsbaum and Robert L. Glass, Measuring & Motivating Maintenance Programmers, Prentice Hall, New Jersey, 1992.
- [MARCINIAK90] Marciniak, John J., and Reifer, Donald J., Software Acquisition Management: Managing the Acquisition of Custom Software Systems, John Wiley & Sons, Inc., New York, 1990.
- [MARSH90] Marsh, Alton, "Pentagon Up Against a Software Wall," *Government Executive*, May 1990.
- [McGIBBON96] McGibbon, Thomas, *A Business Case for Software Process Improvement: A DACS State-of-the-Art Report*, Data & Analysis Center for Software, Rome Laboratory, Rome, New York, September 30, 1996.
- [McLURE98] McLure, David, as quoted by Edward Cone, "Crash-Landing Ahead," *InformationWeek*, January 12, 1998.
- [MOSEMANN95] Mosemann, Lloyd K., II, Keynote Address presented to the 1995 Software Technology Conference, Salt Lake City, Utah, 1995.
- [MULLINAX98] Mullinax, Don, as quoted by Edward Cone, "Federal CIOs Look Past Failures," *InformationWeek*, January 12, 1998.
- [PAT92] "Software Process Action Team Final Report: Process Improvement for Systems/Software Acquisition," Air Force Systems Command, June 30, 1992.

- [PAULSON79] Paulson, Paul J., as quoted in the *New York Times*, May 4, 1979.
- [PIETRUCHA97] Pietrucha, Bill, "US Infrastructure Vulnerable to Computer Attack — Report," *Newsbytes*, October 24, 1997.
- [PRESSMAN92] Pressman, Roger S., *Software Engineering: A Practitioner's Approach*, Second Edition, McGraw-Hill, New York, New York, 1992.
- [RAND93] "An Analysis of Weapon System Cost Growth," RAND Corporation, MR-291-AF, 1993.
- [RICHARDS90¹] Richards, Evelyn, "Society's Demands Push Software to Upper Limits: More Computer Crises Likely," *The Washington Post*, December 9, 1990.
- [RICHARDS90²] Richards, Evelyn, "Pentagon Finds High-Tech Projects Hard to Manage: The Army Still Awaits Computerized Battlefield," *The Washington Post*, December 11, 1990.
- [SANTAYANA05] Santayana, George, Chapter 12, "Reason in Common Sense," *The Life of Reason*, Volume 1, 1905.
- [SILVESTRI97] Silvestri, George T., "Employment Outlook: 1996-2006; Occupational Employment Projections to 2006," *Monthly Labor Review*, U.S. Department of Labor, November 1997.
- [SWOPE50] Swope, Herbert B., U.S. Journalist Speech, as quoted in *The Columbia Dictionary of Quotations*, Columbia University Press, 1995.
- [VESSEY84] Vessey, GEN John W., as quoted in the *New York Times*, February 25, 1984.
- [YOURDON97] Yourdon, Edward, *Death March: The Complete Software Developer's Guide to Surviving "Mission Impossible" Projects*, Prentice Hall PTR, Upper Saddle River, New Jersey, 1997