Learning Track 6: Software Vulnerability and Reliability

In addition to disasters, viruses, and security breaches, defective software and data also pose a constant threat to information systems, causing untold losses in productivity. An undiscovered error in a company's credit software or erroneous financial data can result in losses of millions of dollars. Table 1.1 illustrates just of a few of the better known system quality problems of 2012.

TABLE 1-1 Top Software Glitches

Software glitch	Description
Toyota Motors Corporation	Recall of 2 million cars to fix a software glitch that caused the cars to stall.
United Airlines	5,000 flights grounded due to a network router slowdown.
Wall Street Journal	The Journal's web site taken down because its homepage failed to load.
NYSE	The NewYork Stock Exchange suspends trading in all securities for several hours due to a software error.
The NASDAQ stock exchange	The NASDAQ stock exchange computers failed in the first half hour of trading in Facebook's IPO stock offering. Investor orders were not recorded so investors did not know what price they paid for the stock, or even if they owned it.
Royal Bank of Scotland	Customers lost access to their accounts when a system upgrade was performed
LinkedIn passwords leaked	6.6 million passwords leaked online due to a software glitch.
GoDaddy	The Web hosting site had problems in the DNS (Domain Name System) causing thousands of Web sites that it hosts to go down.
Knight Capital	Knight lost $440 million in trading due to a software glitch which generated erroneous buy orders for major companies' stock. Knight closed it doors when it could not make good on the orders.
Apple Map errors	Apple prematurely released its Map app with many bugs like no labels, wrong directions, and no public transportation records.

BUGS AND DEFECTS

A major problem with software is the presence of hidden bugs or program code defects. Studies have shown that it is virtually impossible to eliminate all bugs from large programs. The main source of bugs is the complexity of decision-making code. Even a relatively small program of several hundred lines will contain tens of decisions leading to hundreds or even thousands of different paths. Important programs within most corporations are usually much larger, containing tens of thousands or even millions of lines of code, each with many times the choices and paths of the smaller programs. Such complexity is difficult to document and design—designers

continued

may document some reactions incorrectly or may fail to consider some possibilities. Studies show that about 60 percent of errors discovered during testing are a result of specifications in the design documentation that were missing, ambiguous, in error, or in conflict.

Zero defects, a goal of the total quality management movement, cannot be achieved in larger programs. Complete testing simply is not possible. Fully testing programs that contain thousands of choices and millions of paths would require thousands of years. Eliminating software bugs is an exercise in diminishing returns because it would take proportionately longer testing to detect and eliminate obscure residual bugs. Even with rigorous testing, one could not know for sure that a piece of software was dependable until the product proved itself after much operational use. The message? We cannot eliminate all bugs, and we cannot know with certainty the seriousness of the bugs that do remain.

THE MAINTENANCE NIGHTMARE

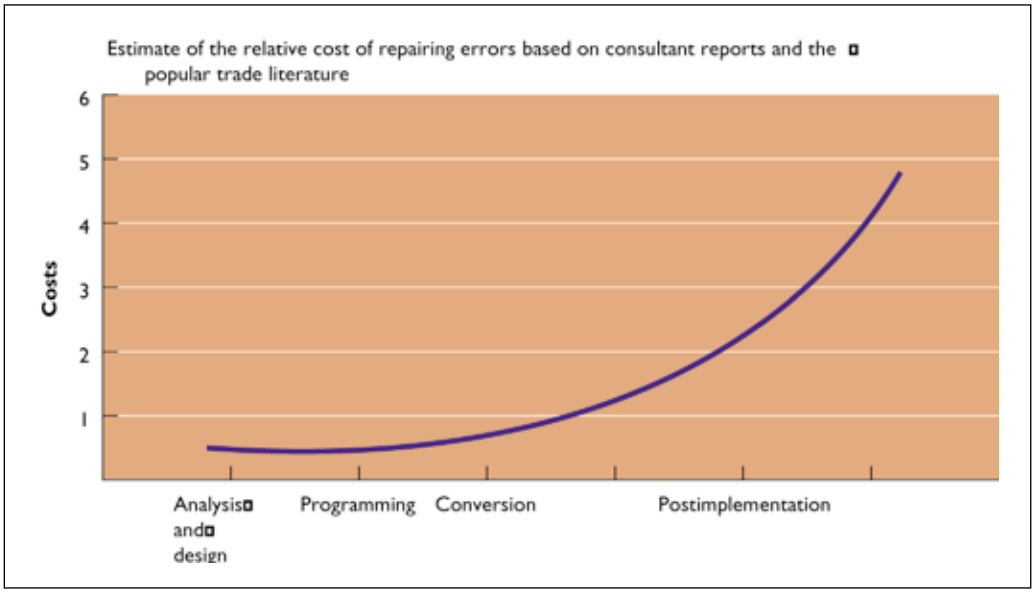
Another reason that systems are unreliable is because computer software traditionally has been a nightmare to maintain. Maintenance, the process of modifying a system in production use, is the most expensive phase of the systems development process. In most organizations nearly half of information systems staff time is spent maintaining existing systems.

Why are maintenance costs so high? One major reason is organizational change. The firm may experience large internal changes in structure or leadership, or change may come from its surrounding environment. These organizational changes affect information requirements. Another reason appears to be software complexity, as measured by the number and size of interrelated software programs and subprograms and the complexity of the flow of program logic among them. A third common cause of long-term maintenance problems is faulty systems analysis and design, especially analysis of information requirements. Some studies of large TPS systems by TRW, Inc., have found that a majority of system errors—64 percent—result from early analysis errors.

Figure 8-1 illustrates the cost of correcting errors based on the experience of consultants reported in the literature. If errors are detected early, during analysis and design, the cost to the systems development effort is small. But if they are not discovered until after programming, testing, or conversion has been completed, the costs can soar astronomically. A minor logic error, for example, that could take 1 hour to correct during the analysis and design stage could take 10, 40, and 90 times as long to correct during programming, conversion, and postimplementation, respectively.

continued

FIGURE 8-1 The Cost of Errors Over the Systems Development Cycle.



The most common, most severe, and most expensive system errors develop in the early design stages. They involve faulty requirements analysis. Errors in program logic or syntax are much less common, less severe, and less costly to repair than design errors.

RESOURCE ALLOCATION DURING SYSTEMS DEVELOPMENT

Views on resource allocation during systems development have changed significantly over the years. Resource allocation determines the way the costs, time, and personnel are assigned to different phases of the project. In earlier times, developers focused on programming, with only about 1 percent of the time and costs of a project being devoted to systems analysis (determining specifications). More time should be spent in specifications and systems analysis, decreasing the proportion of programming time and reducing the need for so much maintenance time. Documenting requirements so that they can be understood from their origin through development, specification, and continuing use can also reduce errors as well as time and costs. Current literature suggests that about one-quarter of a project's time and cost should be expended in specifications and analysis, with perhaps 50 percent of its resources allocated to design and programming. Installation and postimplementation ideally should require only one-quarter of the project's resources. Investments in software quality initiatives early in a project are likely to provide the greatest payback.

SOFTWARE METRICS

Software metrics can play a vital role in increasing system quality. Software metrics are objective assessments of the system in the form of quantified measurements. Ongoing use of metrics allows the IS department and the user to jointly measure the performance of the system and identify problems as they occur. Examples of software metrics include the number of transactions that can

continued

be processed in a specified unit of time, online response time, the number of payroll checks printed per hour, and the number of known bugs per hundred lines of code.

For metrics to be successful, they must be carefully designed, formal, and objective. They must measure significant aspects of the system. In addition, metrics are of no value unless they are used consistently and users agree to the measurements in advance.

TESTING

The stages of testing required to put an information system into operation are program testing, system testing, and acceptance testing. Early, regular, and thorough testing will contribute significantly to system quality. In general, software testing is often misunderstood. Many view testing as a way to prove the correctness of work they have done. In fact, we know that all sizable software is riddled with errors, and we must test to uncover these errors.

Testing begins at the design phase. Because no coding exists yet, the test normally used is a walkthrough—a review of a specification or design document by a small group of people carefully selected based on the skills needed for the particular objectives being tested. Once coding begins, coding walkthroughs also can be used to review program code. However, code must be tested by computer runs. When errors are discovered, the source is found and eliminated through a process called debugging.

Electronic commerce and electronic business applications introduce new levels of complexity for testing to ensure high-quality performance and functionality. Behind each large Web site, such as Amazon.com, eBay, or E*TRADE, are hundreds of servers, thousands of miles of network cable, and hundreds of software programs, creating numerous points of vulnerability. These Web sites must be built and tested to make sure that they can withstand expected—and unexpected—spikes and peaks in their loads. Both Web site traffic and technical components, such as hardware, software and networks, must be considered during application development and during testing.

To test a Web site realistically, companies need to find a way to subject the Web site to the same number of concurrent users as would actually be visiting the site at one time and to devise test plans that reflect what these people would actually be doing. For example, a retail e-commerce site should create a test scenario where there are many visitors simply browsing and some making purchases.

Testing wireless applications poses additional challenges. Many wireless and conventional Web applications are linked to the same back-end systems so the total load on those systems will increase dramatically as wireless users are added. Automated load testing tools that simulate thousands of simultaneous wireless Web and conventional Web browser sessions can help companies measure the impact on system performance.

Many companies delay testing until the end of the application development phase, when design decisions have been finalized and most of the software program code has been written. Leaving Web site performance and scalability tests until the end of the application development cycle is

continued

extremely risky because such problems often stem from the fundamental workings of the system. To minimize the chance of discovering major structural problems late in the system's development process, companies should perform this testing well before the system is complete. This makes it possible to address performance bottlenecks and other issues in each application level or system component before everything is integrated.