

Software Quality Matters

Newsletter of The Software Quality Institute at The University of Texas at Austin in Cooperation with the Austin Software Council and the Austin Software Process Improvement Network

Volume 6, Number 3



Fall 1998

Part I

Software Reliability—Keeping the Bugs Out

by Allen Johnson



Since the early days of computers, keeping bugs out of the software has been a concern. Over the years, the reliability of the hardware and software has increased dramatically. Even with this dramatic increase, reliability has not kept up with the increase in complexity and the importance the customer places on it. Now that everything we do seems to depend on some type of computer and its software—from the life-sustaining devices in hospitals to the airplanes we fly in and the cars we drive—more reliable software has become an economic necessity.

The proliferation of computers has also generated more and more novice users. Software must be more robust to withstand the onslaught of these novices.

Costs of Quality

Anyone in the business of developing software should understand the costs that are associated with quality. This understanding will help them make rational decisions about what strategy to employ to keep bugs out and how much effort to expend on this activity. In a nutshell, if you take action, there is cost; if you take no action, there is more cost.

Farewell to Herb Krasner Welcome to Gary Cobb

by Cinda Cyrus

After overseeing its startup and leading UT's Software Quality Institute for five years, Herb Krasner has stepped down as Chair of the SQI Advisory Group.

Gary Cobb, of Dell, is the new Chair. According to Joanne Click, SQI's director, Cobb has made substantial contributions to SQI from the beginning. He has served as a SQI seminar instructor, been a contributor to the Software Project Management Program, and more recently co-authored the new software quality award criteria for the Greater Austin Quality Council.

Krasner's Contributions

"Without a doubt," Click said, "Herb Krasner's leadership is largely responsible for SQI's steady growth and success in its first five years. Krasner put SQI on the map."

Krasner/Cobb (continued on p. 5)

The cost of quality includes more than the costs of prevention, appraisal, litigation, and maintenance. There are also the costs of project rework, lost management time, rerun, lost credibility, lost business and goodwill, lost opportunity, lost assets, and perhaps more.

Defining Reliability and Bugs

To understand the relationship between the removal of bugs or avoiding bugs and software reliability, it is helpful to define reliability in street language, or layman's terms. Software reliability can be defined as the odds that a piece of software will work wonderfully for a user without some bug causing a foul up in the operation. (This assumes, of course, that the user is not misusing or abusing the software.) We know how to measure this attribute, and we have various models to help us predict this reliability.

After using *bug* in the preceding definition, I have to say that *bug* is not really a good technical term. A better term is *problem*, or *anomaly*. We can define *problem* as a deviation from specifications or expected results.

Problems come in different flavors, depending on when the problem is found. If the problem is found in a later phase or process than where it was introduced, then we can call it a defect. If the problem is found in the current phase or process, then we can call it an error.

Another more formal term for defect is *fault*. A fault resides in the resources used in a process and is viewed as an input to a process and represents an incorrect state or condition of the system to which the process belongs. Then we have the often dramatic event where the faulty resource is used by a process or system and causes an error in the output, which is observed; this event is a failure. If we can somehow keep a fault from resulting in a failure when the resource is used, we can say that we have a

Bugs (continued on p. 2)

What's Inside

Fall SQI Seminars	3
UT's MSE Program	5
Legal Implications of Unsuccessful Software Projects	6
SQI Calendar	8

DEVELOP NEW SKILLS AND ENHANCE CURRENT ONES WITH SQI SEMINARS, page 3

Bugs (continued from page 1)

fault-tolerant system. This is most beneficial when the consequences of failure are very expensive or life threatening.

Where Do Bugs Come From?

What is the origin of these bugs that cause software failures? Schneiderwind and Hoffman came up with the following possible origins for these bugs.

- Design defects/errors (system and software)
- Coding defects/errors
- Clerical mistakes
- Debugging inadequacies
- Testing mistakes

Where do these bugs originate? Capers Jones collected the data on the origin of defects shown in Table 1.

Table 1. Origin of Bugs

Defect/Origin	Defects per Function Point
Requirements	1.00
Design	1.25
Coding	1.75
Documentation	0.60
Bad fixes	0.40
Total	5.00

If we are to come up with effective ways of eliminating defects, we must understand their origin so that we can eliminate their cause. Function points are a function-oriented software metric that enables an organization to estimate the amount of effort it will take to produce the software.

After looking at where defects originate, we need to look at how they show up. Some examples follow.

- Memory accesses out of your program's space
- Writing over code within your program
- Overwrites of your code by an external driver
- Runaway programs
- Heap overruns
- Library routine overruns of all local arrays
- Leaving blocks unfreed (memory leaks)

All bugs are not created equal in the eyes of the user. Boris Beizer has come up with the following effective classification of the consequences of faults when they cause a failure.

- *Mild*—symptoms offend us aesthetically
- *Moderate*—outputs misleading or redundant
- *Annoying*—causes dehumanizing system behavior (e.g., money machine refuses to cash your paycheck)
- *Very serious*—instead of losing your paycheck, the system credits it to another account
- *Extreme*—problem not limited to a few users
- *Intolerable*—long-term, unrecoverable corruption of database
- *Catastrophic*—decision to shut down taken out of our hands; system fails
- *Infectious*—corrupts other systems even though it does not fail itself

Keeping Bugs Out

Since bugs can have a terrible impact on a system, we need to put a strategy in place to keep bugs out. The solution is to implement a strategy that promotes the following.

- Don't put 'em in!
- Find 'em fast and stomp 'em out fast!
- Keep track of the little critters.
- Like breakfast cereal, determine how many of the little critters you can leave in without scaring off the customers.
- For those you can't keep out but can't afford the consequences of their causing a failure, figure out how to tolerate them.

How do you *not* put bugs in in the first place? A disciplined development process employing software process improvement leads to the elimination of most bugs. If the organization is immature, members can start with basic stuff like introducing metrics, inspections, configuration management, and defect tracking. In fact, the organization can make great strides by implementing the CMM KPIs that will get it to Level 2. This means putting in place good project management, project planning, configuration management, and software quality assurance.

If removing defects is really critical, then the group needs to continue improving the process to get it to CMM Level 5 if possible but at least to Level 3. Getting to Level 3 means putting in place training, technical practices such as inspections and testing, and focusing on standards for their processes. If the organization is dealing with a rather static environment, it may want to consider adding approaches such as cleanroom to its arsenal. If it needs to go further to Level 4, then it needs to add process measurement, process analysis, and quantitative quality plans.

To really get into the process improvement cycle means going to Level 5 (Optimizing). This means the organization becomes more proactive in evaluating and implementing new technology and performing defect prevention. With an experienced creative team, further improvements are somehow always justified. The key is to keep improving processes until the defects are within an acceptable level and where return on investment does not justify going any farther.

Implementing Software Reliability Engineering

When you get down to it, the real issue from the customer's perspective is reliability and integrity—not removing defects. This can be a significant cost that is difficult for the customer to track. The increasing cost in this area is reflected in the explosion of demand for help desk services. This means planning and measuring the reliability of the product in every phase so that when it comes time to ship the product, it meets the customer's reliability requirements.

For the development organization that is serious about delivering a reliable product, implementing software reliability engineering (SRE) methodology into its development life cycle should be considered.

SRE involves the following basic activities.

- Define an operational profile based on customer modeling.
- Generate test cases automatically based on frequency of use reflected in the operational profile.
- Deliver software in increments to system test with quality factor assessments.
- Employ cleanroom development techniques with random testing based on the operational profile.
- Test to reliability objectives.

Bugs (continued on p. 3)

SW Professionals Develop New Skills And Enhance Current Ones with SQI Seminars



SQI seminars this fall offer software professionals the opportunity to develop new skills and enhance current ones.

Ranging in length from one to five days, each course is designed to provide real knowledge immediately applicable in the workplace. Following are the courses currently on the schedule.

Object-oriented Design and Programming in C++ and Java for C Programmers

Glenn Downing, instructor

Taught two mornings per week for five weeks, from Sept. 15 to Oct. 22, this course presents fundamentals of C++ and Java, with special attention to developing well-defined OO programs. Four out-of-class C++ programming projects requiring from 10 to 25 hours to complete are required. This course is designed for software developers. A working knowledge of C arrays and pointers is essential.

Capability-based Assessments for Software Organizations

Gary Cobb, instructor

This two-day seminar, held Sept. 24-25, describes the assessment process, the organization and project responsibilities for fulfilling Levels II-V, and the best approaches to tracking continuous improvements. This course prepares software professionals and their managers for an SEI assessment.

People-Capability Maturity Model

Bill Curtis, instructor

This one-day seminar, held Sept. 25, details SEI's People Capability Maturity Model (P-CMM) and presents guidelines for using the model to address the human resource issues involved in improving software organizations. The course is designed for members of software engineering process groups, software managers, human resource specialists in high tech organizations, and software or information system developers interested in better management of talent within their organizations.

Practical In-Process Inspections

Geree Streun, instructor

This two-day seminar, held Sept. 28-29, offers an in-depth look at the in-process inspection technique, an advanced technique designed to find defects in each and every software development process artifact. The class provides vigorous hands-on experience using inspection techniques. The intended audience includes software managers and engineers and software quality assurance personnel.

Software Scheduling and Cycle Time Reduction

Dennis Frailey, instructor

This one-day course, held Sept. 30, shows how to develop a realistic software project schedule and how to reduce the schedule with cycle time improvement techniques. Techniques include pert, Gantt, and network

charts and critical chain/critical resource analysis. Exercises apply these techniques to real problems. Students should be familiar with the basic steps of the development process.

Constructing a Software Requirements Specification

Cecil Martin, instructor

This two-day seminar, offered Oct. 5-6, presents a step-by-step process for developing a software requirements specification (SRS) using data and structural and behavioral models produced from analyzing the problem. Course information includes the characteristics of a good SRS, how to construct an SRS, and how to validate an SRS. Attendees should be experienced in developing and maintaining software.

Object-oriented Development Using the Unified Modeling Language

Charles Richter, instructor

This five-day seminar, offered Nov. 30-Dec. 4, begins with an introduction to basic OO concepts and design principles and to basic concepts and notations for static and dynamic modeling and design. These topics are followed by discussions of more advanced concepts and UML mechanisms for extending the notation, design heuristics, system architecture, class design, and frameworks and design patterns. No knowledge of OO concepts or of OO development is assumed.

Seminars (continued on p. 4)

Bugs (continued from page 2)

In each phase of the development life cycle, several important activities contribute to the successful SRE performance. In the feasibility and requirements phase, for example, one activity is to identify customer reliability needs. In the design and implementation phase, one important activity is to engineer the product to meet the reliability objectives because you are not going to be able to test the reliability into the product. You have to design reliability into the product.

During the system test and field trial phase, one activity involves building on the knowledge accumulated from previous phases about the allocation of functions to software modules and the anticipated usage by the customer of those functions to determine an operational profile. Because of the diversity of users and the dynamic nature of user experience over time, this can be a difficult challenge. It is a worthwhile activity because, at the very least, it should improve the knowledge available concerning what is important to users. With an operational profile established, then effective reliability growth testing can be done.

Actually, the gathering of data concerning reliability growth should begin in the requirements phase, but most organizations choose to start in the system test phase. Finally, after shipping the product, reliability should be monitored to see how the product performs against the objectives established. Also, customer satisfaction with reliability should be tracked.

If you are going to grow reliability, it is important to understand the factors that influence product reliability so that you know what needs to be controlled and how to prioritize the decisions you make about removing defects. The three factors that influence reliability are when and how faults are introduced, when faults are removed, and the operational profile. The strategy is to put the removal of faults as close as possible to the point when they are introduced. This is when cost is minimal and the probability of making a good fix is maximized.

The operational profile will help you understand which software components will have the greatest impact on customer-perceived reliability if they fail. This will help you make decisions

Bugs (continued on p. 4)

Seminars (continued from page 3)

Software Testing

Gary Cobb, instructor

This three-day seminar, offered Oct. 21-23, covers the role of software testing throughout the total life cycle of the typical software development process. The course is designed for integration and test engineers, software quality engineers, software engineers, software systems engineers, software managers, and chief systems engineers.

Constructing a Software Design Specification

Cecil Martin, instructor

This two-day seminar, offered Nov. 9-10, presents a methodology for constructing a software design specification (SDS). Preliminary and detail design processes will be covered. Students should be experienced in developing and maintaining software.

Object-oriented Concepts and Approaches

Charles Richter, instructor

This one-day seminar, offered, Nov. 23, explores basic object technology and outlines the steps involved in OO development and the different forms of reuse of OO designs and code. No knowledge of OO concepts or of OO programming is assumed.

Java as an Object-oriented Programming Language

Charles Richter, instructor

This one-day seminar, offered Nov. 24, is a crash course describing the Java language and its use for OO programming. It covers basic concepts such as classes and inheritance, as well as more advanced topics such as threads and exceptions. The course is

designed for those who want a quick-paced, intensive discussion of Java's basic features and how Java differs from C++. A familiarity with C is helpful; knowledge of OO concepts is helpful but not required. Knowledge of C++ or Smalltalk is not required.

Setting up a Metrics Process

Gary Cobb, instructor

This two-day seminar, offered Dec. 10-11, provides a process and examples for controlling a software development project to a plan through the selection and use of software metrics. The purpose of the course is to make all team members aware of the way metrics should be used to guide team decision, as well as to self-manage their development processes. The course is designed for software systems engineers, software quality engineers, software engineers, and software project managers.

Object-oriented Programming in Java

Charles Richer, instructor

This five-day course, offered Dec. 14-18, covers the fundamentals of programming in Java. Both the language and some of the standard packages are covered, and numerous exercises and laboratories will provide hands-on experience. The course is designed for those wanting to write applications or applets in Java.

Additional Information

To learn more about these courses or other SQI programs, contact—

Software Quality Institute

Phone: (512) 475-6779 or

1-800-687-8012

Fax: 512-471-4824

E-mail: elms@sqi.utexas.edu

Software Quality Matters

Editor: Cinda L. Cyrus

© 1998 by The University of Texas at Austin. All rights reserved

Printed in the United States of America

Editorial Board Members

Al Dale	Cinda Cyrus
Jack Odum	Herb Krasner
Joanne Click	

Contributors to this Issue

Candy Walser Berry	Gale Hathcock
Debbie Caples	Herb Krasner
Joanne Click	Marilyn Robertson
Cinda Cyrus	Elaine Stanfield
Carolynn Elmsphauser	

Software Quality Matters is published quarterly by the Software Quality Institute at The University of Texas at Austin College of Engineering, in cooperation with the Austin Software Council and the Austin Software Process Improvement Network.

Brand names, company names, trademarks, or other identifying symbols appearing in text and illustrations are used for educational purposes only and do not constitute an endorsement by the authors, editor, or publisher.

For additional information, contact —

Software Quality Institute
The University of Texas at Austin
PRC MER Mailcode R9800
Austin TX 78712-1080
Phone: (512) 471-4875
or: 1-800-687-8012
Fax: (512) 471-4824
E-mail: info@sqi.utexas.edu
<http://www.utexas.edu/coe/sqi>

Bugs (continued from page 3)

about where to spend the most effort testing. According to Musa, Iannino, and Okumoto, the operational profile is the "set of relative frequencies of occurrence of the run type (user function), usually expressed as fractions of the total set of runs (probabilities)."

Using Testing Effectively

In order to use testing effectively to remove bugs, take the following steps. Use the operational profile to focus on testing high-use modules. Take advantage of any building block components of the software product to avoid overtesting high-use components because they are used again in testing other components. Use a disciplined process to achieve maximum coverage with the least effort. View your testing time and testers as a finite resource that must be conserved through judicious use.

Use complex test scenarios (approximation of real-life situations that the customer may create) to help reduce the number of difficult problems that typically escape to the field and are costly to diagnose, re-create, and fix successfully. Most often, testing groups will only use simple tests and then wonder why reliability in the field is so much worse than what they measured during test. Set a target of

achieving at least 10 percent better reliability in test than what is required in the field to compensate for the fact that you can never think of all the complex ways the customer actually uses the system.

Part II

In the second part of this article, which will appear in the next edition of *Software Quality Matters*, the author will discuss improving system reliability, growing reliability, testing reliability growth, implementing fault tolerance, and characteristics of universal fault tolerance.

About the author. Allen Johnson, Ph.D., is the CEO of Rainbow Analysis Systems Group, a faculty member for SQI, an advisory board member for SQI, and a board member for the Greater Austin Quality Council. He has taught graduate courses in software engineering and fault-tolerant computing in the Electrical and Computer Engineering Department at UT. He is a retiree from IBM where he helped implement reliability, availability, and serviceability into many of the computer systems, including System 360, System 370, and RS/6000. Johnson has more than 30 publications and patents.

Note: for references, please contact Johnson at allen@rasgroup.com

UT's Executive Software Engineering Graduate Program Geared toward Full-Time Workers

byCarolynn Elmshaeuser



UT's Executive Software Engineering graduate program, known as MSE, is looking forward to another exciting fall semester. MSE offers students the opportunity to earn a Master's degree in engineering in two years while continuing to work full time.

Twenty-two of the 46 students currently enrolled in the program are on track to graduate in December. These students represent a total of 23 companies. The companies represented include 3M, Applied Research Labs, BMC Software, Cisco Systems, Compaq, Comsys, Digitech Systems, Fisher Rosemount Systems, Georgetown

Engineering, GlobeSet, IBM, LSI Logic, Motorola, National Instruments, Objective Resources Group, Pavillion Technologies, Pioneer Natural Resources, Schlumberger, Tandem, Tivoli, Tracor, Trajecta, and Wayne-Dresser.

Student Requirements

MSE students meet all day Friday and all day Saturday one weekend per month during the fall and spring semesters, and they take an independent study/research course during each summer of the program. They also attend a week-long seminar each January and complete a master's report.

Students in the MSE program meet all University requirements of traditional students and are taught by the same instructors as traditional students. The instructors are members of the faculty of UT's Department of Electrical and Computer Engineering.

New Class Starts January

The MSE program admits new students each January.

To obtain more information on this program, please call (512) 475-6779, or send an e-mail request to elms@sqi.utexas.edu. Applications are still being accepted for January 1999 admission.

Krasner/Cobb (continued from page 1)

Al Dale, SQI's executive director, introduced Krasner as "my co-investigator" in seeking guidance from the CEOs of Austin's major industries in 1993 when SQI was in its initial stages of development. The goal of those early visits was to determine industry's greatest need that SQI might address. The CEO responses all turned to quality of software issues, which has been SQI's focus since the outset.

Krasner has made many other contributions to SQI, Click said.

"One of Herb's stellar assets, in my view, is his vision of community," Click added, "the vision of a group of professionals working together collaboratively for the good of the whole." His good nature and fun-loving spirit are contagious. He's creative, enthusiastic, loves to see things happen—he makes others want to be involved."

Early on, Krasner persuaded the American Society for Quality (ASQ) to hold its Fifth International Conference on Software Quality (5ICSQ) in Austin. He served as general chair for a legendary event that has gone down in history as among the best-ever software quality conferences. That event put SQI on the map. Click said. It gave the local community an opportunity to learn more about what quality means and to meet the world's software quality leaders. The conference also gave Austin the opportunity to shine.

Krasner has written more articles, all of them highly insightful, for *Software Quality Matters* than anyone else, a major feat in itself, Click said. He has also chaired almost every SQI quarterly Advisory Group meeting and attended nearly every meeting of the Austin Software Process Improvement Network (A-SPIN). Krasner founded A-SPIN prior to SQI.

Click added that Krasner has also been a contributor to the startup of the SQI Series with Macmillan, co-instructed (with Bill Curtis) SQI's first seminar, and has been a strong supporter of the Software Project Management Program.

"Herb will continue to be involved with SQI, and we are looking forward to continuing our productive working relationship and to seeing more of his articles in *Software Quality Matters*. He does love to keep us on our professional toes," Click added with a laugh.

Cobb's Background

Cobb has been directly involved in software development for 25 years in both industry, at TI, Lockheed, and Dell Computer Corporation, and in academia at The University of Texas at Austin, Embry-Riddle, and Southwest Texas State University. He worked for several years developing a software developers toolkit for testing, including a compiler preprocessor to instrument source code in order to measure completeness of path testing.

As a systems engineer, Cobb documented and tested the feasibility of user requirements. He has worked in object-oriented development environments that make the job of testing easier. He also had the unusual opportunity to employ theorem proving tools to an air traffic control problem involving a collision avoidance algorithm, developed at Lincoln Laboratories

Currently employed at Dell, Cobb is manager of the Millennium Program Office for Compliance/Certification in Information Technology where he is responsible for ensuring that the software that Dell runs its business on will survive the transition into the next century.

"My auxiliary job," Cobb says, "is described as 'on the graduate faculty of the Department of Computer Science at Southwest Texas State University.' I teach two courses at SWTSU: a senior-level course, titled Introduction to Digital Multimedia, and a graduate-level course, titled Multimedia Systems Design."

In June, Cobb served as local chair for the IEEE International Conference on Multimedia Systems and Computing.

Cobb is also on the board of directors of the Austin Software Process Improvement Network (A-SPIN) and of the Lake Travis Education Foundation. He is also an examiner for the Austin Quality Council's Greater Austin Quality Award.

Wants Your Input

"If you have questions about SQI or suggestions for its future direction," Cobb says, "don't hesitate to give me a call at 728-4693 (work) or 261-6315 (home)." I'm looking forward to this new role, and I'm looking forward to its challenges," Cobb concluded.

Legal Implications of Unsuccessful SW Projects

by Herb Krasner

(Disclaimer. This article should not be construed as providing any legal advice or a legal opinion, and it does not purport to give a summary of the law or a comprehensive analysis of the legal issues raised therein. Do not act or rely upon legal information or views in this article without consulting an attorney.)



Part I of this article examined the nature of legal actions in two-party software development situations. Now, in Part II, the author suggests actions software professionals can take to mitigate litigation

The questions of interest to software professionals are these.

- What can be done to reduce the risk of litigation?
- What records should be kept in case a lawsuit should occur?
- What should be done to reduce the amount of exposure should litigation occur?

The easiest way to avoid the risk of litigation, of course, is to have a successful project that is done well, meets its objectives, and satisfies its customers. Unfortunately, this currently happens much too infrequently and is easier said than done. Most of the advice that I would give about being more successful is fully consistent with creating valuable evidence. If the right things are done, no additional effort is required.

Individuals at various organizational levels of responsibility can take some preventative steps. These ideas should be treated as a starter set of guidelines or, perhaps, as rules of thumb, and not as legal advice for any specific situation. I give this advice from the perspective of a subject matter expert who has had difficulties trying to re-create, reconstruct, or assemble information that was not available or developed during the actual project but was absolutely needed to prepare for a lawsuit.

These suggestions have a process and metrics focus because there is nothing an opposing lawyer would love more than to be able to expose that the development organization had an arbitrary or unreasonable process.

Arbitrary notionally means *by chance*. Other synonyms for arbitrary include random, off-hand, unscientific, subjective, inconsistent, erratic, capricious, and irresponsible. *Unreasonable* notionally means *without a logical basis*. Synonyms include thoughtless, nonsensical, foolish, and fallacious.

If your project's process was by chance and/or illogical, then you could be in big trouble. The underlying themes of my suggestions deal with achieving the opposite of these two concepts and being able to show that in court. This involves:

- reasoned decision-making,
- behaving in a professional manner,
- defining and using processes,
- measuring and analyzing factual data, and
- implementing effective and commonly known best practices.

Software and Database Engineers

Lawsuits sometimes occur when projects fail due to the technical complexities of the system, but they occur more often because of incompetence, poor personal planning, and lack of discipline on the part of software and database engineers.

Following are several steps software and database engineers can take to avoid being caught up in an unsuccessful project and subsequent litigation.

1. Behave according to the IEEE/ACM Software Engineering Code of Ethics. (See *IEEE Computer* or *CACM*, November 1997.) Perform the development work in a professional manner.
2. Take the SEI-developed Personal Software Process (PSP) training (which SQI offers, by the way) so that your own personal software process is disciplined.
3. Seek out certification programs for software engineers that will be emerging during the next decade. In the state of Texas this certification has recently been approved through the State Board of Professional Engineers.
4. Make sure you are qualified to do a particular job, and if you are not fully qualified then document the areas in which you are not, for the record.
5. Decline to accept a position on a project that appears preconditioned to fail. Put a disclaimer in your personnel file if you must, and quit if you have to. This is actually a win-win situation for your company as well if it

can keep the company from starting projects that will fail. Then resources can be given to projects that will succeed.

6. Use rigorous, defined, and measurable personal, team, and project software processes for developing software systems. This will help prevent negligence suits as well as improve performance.
7. Contribute specific performance data to the organization's software measurement database.
8. Perform a rigorous estimation process when planning and bidding your portion of a new project, using effort and time ranges and confidence levels. Document the unknowns, the risks, and your assumptions, and do not sign off on estimates that management has changed without your agreement. Do document that it occurred.
9. Purchase professional liability insurance if you are not covered by your company. Errors and Omissions insurance policies for software engineers exist.
10. When problems occur that jeopardize the stated success goals of the project, document them and report them immediately. Small problems become big problems more quickly when hidden.

Software Project Managers

The pressure is on the project manager to make success happen, and the project manager is the person most likely to be named in a lawsuit. Lawsuits can occur when projects fail due to poor project planning and management. This occurs when the project manager establishes flawed project goals; signs up to overly optimistic (best-case) estimates; makes unreasonable delivery commitments; wimps out; allows the project scope to change uncontrollably; looks for miracles; and leaves no contingency room to negotiate, prioritize, or deal with risks.

Software project managers should do the following to avoid being pinned with a project failure.

Legal (continued on p. 7)

Legal (continued on page 6)

1. Become an expert in software project planning and management techniques. If you are newly assigned to this role, then become a certified software project manager. (SQI's next class starts March 16, 1999.)
2. Make sure you know what project success means. A written charter statement signed off by management helps.
3. Estimation is where most project managers fall down. A poor estimation process leads to poor commitments and perhaps differing expectations for success.
4. Do not sign up to run a fixed-price, fixed-deadline project without a written understanding with the customer that the requirements to be implemented are negotiable and will be based on what is learned during the development process.
5. Make sure that at least one degree of freedom (scope, effort, quality, and schedule) is available for negotiations and that planned contingency resources are in place
6. Adopt a mature development process for your project, at least CMM Level 3 and preferably CMM Level 4.
7. Rigorously measure the size and scope of the software development project. Also measure and track the changes that occur to them.
8. Along with your chief programmer, rigorously analyze requirements and then track the progress toward meeting them. Trace their implementation all the way into code and test.
9. Measure and track the size and effort for the requirements and their realizations in design, code, test, data redefinitions, and so forth.
10. Use formal inspection or formal review techniques for project plans, requirements, designs, code, test plans, test cases, and so forth.
11. Hold regular progress review meetings with upper management. Discuss and document problems, issues, risks, assumptions, uncertainties, action items, decisions, and barriers to success. Don't hide significant problems because you want to manage your way out of them.

Business Unit & Senior Management

Poorly defined projects are much more likely to fail because of fuzzy goals. *Laissez faire* and trust-me management styles help this happen. Stamp out bad projects before they start.

Some customers are good, and some are not. Many times the key to avoiding a lawsuit is knowing which customers to turn down. Some lawsuits occur, however, due to deteriorating customer relations when senior management is asleep at the wheel. Never assume that no news is good news! Anticipating and dealing with customer problems is what good senior managers do.

“ Poor project planning and management practices can leave project managers with no room to negotiate, prioritize, or deal with risks.”

Following are steps senior managers should take.

1. Evaluate your organization's performance as if it were a subcontractor or vendor against emerging industry standards.
2. Ask good questions and ask them frequently.
3. Be directly and continuously involved in all risky projects. Actively work with the customers. Proactively engage project members (not just the managers) in discussions about risk and the likelihood of success on the project.
4. Use extended cost-of-software-quality analysis to determine the magnitude of the potential legal risk of project failure, and then establish your contingency plans.
5. Warrant your software and live up to your warranties.
6. Educate your customers about what it takes to develop good software.
7. Push software quality standards down to your suppliers and vendors.
8. Teach the legal implications of software project failure in your senior management and software project manager training programs.
9. Make sure you have adequate liability insurance to cover software project failures.
10. Perform software due-diligence audits, along with software capability appraisals in the business units of your organization to ensure that they use disciplined and effective processes and practices.
11. Hire an expert who knows how to implement effective processes and practices in your organization. This will increase the likelihood of successful project performance and protect against lawsuits.

Software Quality Engineers

Some lawsuits arise from software quality problems. Some occur because a quality process was not in place or because quality wasn't a planned objective. Software quality engineers (SQEs) have a role in assisting and assuring that quality software is produced by assuring product and process compliance with standards.

SQEs should take the following steps to help ensure project success.

1. Become a certified software quality engineer (through American Society of Quality).
2. Make sure that an inspection of the project's objectives and project management plan is part of a formal kickoff.
3. Ensure that quality objectives are defined in the project's software development or quality assurance plan.
4. Add specific quality criteria to all product and process audit exit conditions.
5. Rigorously plan and develop testware if you are responsible for testing.
6. Adopt simple software metrics to measure progress (e.g., problem and defect counts).
7. Define software release criteria on the project (using the organizational standard).

Conclusions

From my experience I draw the following conclusions.

- All lawsuits cannot be avoided.
- The customer/developer relationship and effective communication are keys to avoiding some lawsuits.
- The best intentions of the software engineering staff will not make up for bad business decisions or fundamental mismanagement of projects, which are the most common factors in an unsuccessful project.
- The evidence needed to defend against a lawsuit is the same information needed to prevent a lawsuit from happening. This is also the same information that dramatically increases the likelihood of a successful software project.

Legal (continued on p. 8)

SQI Calendar—Meetings and Conferences of Interest

September 24–25, 1998

SQI Seminar
Capability-based Assessments for Software Organizations

Pickle Research Campus, Austin TX

September 25, 1998

SQI Seminar
People-Capability Maturity Model
Pickle Research Campus, Austin TX

September 28–29, 1998

SQI Seminar
Practical In-Process Inspections
Pickle Research Campus, Austin TX

September 30, 1998

SQI Seminar
Software Scheduling and Cycle Time Reduction
Pickle Research Campus, Austin TX

October 5–6, 1998

SQI Seminar
Constructing a Software Requirements Specification
Pickle Research Campus, Austin TX

October 15, 1998

A-SPIN monthly meeting
The Five Responsibilities of Leadership
6:30 to 9:30 p.m.
Pickle Research Campus, Austin TX

October 16, 23, and 30, 1998

SQI Seminar
Software Testing
Pickle Research Campus, Austin TX

November 9–10, 1998

SQI Seminar
Constructing a Software Design Specification
Pickle Research Campus, Austin TX

November 19, 1998

A-SPIN monthly meeting
Reinforcing Desired Behavior
6:30 to 9:30 p.m.
Pickle Research Campus, Austin TX

November 23, 1998

SQI Seminar
Object-oriented Concepts and Approaches
Pickle Research Campus, Austin TX

November 24, 1998

SQI Seminar
Java as an Object-oriented Programming Language
Pickle Research Campus, Austin TX

November 30–December 4, 1998

SQI Seminar
Object-oriented Development Using the Unified Modeling Language
Pickle Research Campus, Austin TX

December 10–11, 1998

SQI Seminar
Setting up a Metrics Process
Pickle Research Campus, Austin TX

December 14–18, 1998

SQI Seminar
Object-oriented Programming in Java
Pickle Research Campus, Austin TX

March 16, 1999

SQI Professional Certification
Software Project Management Program
New 48-week Sequence Begins
5:15 to 8:45 p.m.
Pickle Research Campus, Austin TX

Legal (continued from p. 7)

- The set of skills, capabilities, rigorous processes, practices, and metrics needed to defend against a lawsuit are the same as those needed to increase the likelihood of a successful project.

It would be better for all concerned if customers, development executives, project managers, software engineers, and other stakeholders would engage in more effective discussions about issues, risks, uncertainties, and other barriers to project success.

These discussions would lead to a continual realignment of differing expectations so that no party is surprised by the outcome. The basis for these discussions is a common vocabulary, shared concepts about the success of the project, and mutual trust.

Lawsuits can occur when trust breaks down; therefore, trust-me management strategies are susceptible. Breakdowns occur because critical information is hidden by the parties concerned, usually out of fear. Once fear is removed from the equation, the likelihood of success goes up dramatically.

In general, the intersection of legal issues and software continues to require new understandings, clarification, and definition. An explosion of Year 2000 lawsuits will make this painfully obvious to all.

This article focused on contracted software development projects. For more information about emerging legal issues in consumer product software, see www.badsoftware.com

Copyright © 1998 by Herb Krasner. All rights reserved.



Software Quality Institute
The University of Texas at Austin
PRC MER R9800
Austin, Texas 78712-1080

Nonprofit Org.
U.S. Postage
PAID
Austin, TX
Permit No. 391