PATTERNS IN IT LITIGATION: SYSTEMS FAILURE (1976-2000)

A STUDY BY PRICEWATERHOUSECOOPERS LLP[1]

> Progress, far from consisting in change, depends on retentiveness...Those who cannot remember the past are condemned to repeat it.
>
> *George Santayana,* Life of Reason

ABSTRACT

PricewaterhouseCoopers conducted a survey of litigation involving information technology (IT) projects over a 25-year period, identifying 120 cases that deal with some form of systems failure. A small set of overlapping patterns appears to describe the events in most cases. Organizations involved in either developing or acquiring information technology can reduce their risk of litigation and its impact by recognizing and avoiding these patterns.

INTRODUCTION

Few professions appear to embody the quote above as the history of development projects using information technology: computer hardware, software, data, networks, and all the other bits that go into such systems. That so many IT systems development efforts are inadequate, unacceptable, late or cancelled altogether has become cliché. However, the common factors that lead to IT systems failures have been well documented for over three decades. Classic software engineering works by Brooks, Weinberg, Yourdon, DeMarco, Gilb, Lister, and a host of others have spelled out the core issues that continue to plague IT development efforts in the 21st Century.[2]

It shouldn't be surprising, then, that a survey of litigation involving information technology over the past 25 years likewise shows a remarkable consistency in the events and causes that lead to such legal action. The majority of these cases fall into the "systems failure" category: the computer software and/or hardware sold by one party to another either fails to work acceptably or never works at all. Analysis of the events and causes behind these systems failure cases yields a small set of consistent, repeating patterns. Parties entering into agreements about buying and/or developing information technology—which today means virtually every business or organization—should use these patterns as guideposts and warning flags to avoid the time and expense of litigation.

In this study, we'll lay out the patterns and issues commonly encountered in these cases, then make recommendations on how to reduce the likelihood and impact of such failures and the possible resulting litigation. Note, however, that this paper does not constitute legal advice; for that, see a professional.

---

[1] Bruce F. Webster, now a Principal at Webster & Associates LLC, is the author of the PricewaterhouseCoopers Patterns in IT Litigation: Systems Failure (1976 – 2000) report. He can be reached at (720) 895-1405. See Appendix C, "About the Author", for more details.

[2] See Appendix B, "Selected Books on Software Engineering", for some representative works.

A SURVEY OF IT LITIGATION: SYSTEMS FAILURE

One principal source of information in conducting this survey was the monthly periodical *Computer Law and Tax Reports* ("CLTR").[3] We collected and reviewed data on cases cited in CLTR over a 25-year period beginning in 1976. We used additional sources to identify other cases and to supplement information gleaned from CLTR, including the yearly Overviews of Computer Case Law Developments and searches for computer-related cases on the CCH, Westlaw, and Lexis databases. However, we must point out that such a survey is neither exhaustive nor definitive. Because there are few standard legal keywords for such cases, blind searching for filings is difficult and yields little. Also, our own experience suggests that the majority of these cases settle before coming to trial or even before a summary judgment is issued.

Within these cases, we focused on those that fell under our classification of systems failure, that is, where one party believed that it had not received the promised benefits—in functionality, performance, and/or reliability—from the IT systems received from another party, if indeed those systems were ever delivered. We collected the information on such cases into a custom database. Appendix A contains more information about the database; it also presents some statistics from the database itself.

THE CORE ISSUE: QUALITY

Weinberg defines quality as "value to some person(s)."[4] When it comes to information technology, where one party ("vendor") is to deliver some combination of IT products ("system") to another party ("client"), there are several core quality values that must be addressed:

- **Reliability**: the system must carry out its functions without causing unacceptable errors or having an unacceptable downtime.
- **Performance**: the system must complete its various operations within timespans acceptable to the client.
- **Functionality**: the system must offer sufficient usable features to meet the client's needs.
- **Compatibility**: the system must interact effectively with existing IT systems, including appropriate external systems under the control of other entities.
- **Lifespan**: the system must continue to offer acceptable reliability, performance, and functionality over a sufficient period of time to warrant the cost to the client, including in many cases having the ability to grow with the client.
- **Deployment**: the vendor must deliver and deploy the system, and the client receive its benefits (reliability, performance, and functionality), in a timeframe acceptable to the client.
- **Support**: the system must have the capability to be upgraded and repaired over time.
- **Cost**: the cumulative expense of developing, deploying, upgrading, and maintaining the system must appear to be justified in the eyes of the client.

The key word in these definitions is "acceptable". Most IT systems failure cases surveyed involved claims by the client that one or more of these values were not acceptable for the system in question.

[3] *Computer Law and Tax Reports*. January 1974 – March 2000. Esther Roditti, Esq., editor; New York, NY.

[4] **Quality Software Management: Volume 1: Systems Thinking**, Gerald Weinberg (Dorset House, 1993).

PATTERNS IN IT SYSTEMS FAILURE LITIGATION

The systems failure cases we found fit with little effort into six overlapping patterns.[5] For the sake of simplicity and consistency in describing these patterns, some standard terms will be used:

- Systems – the specific software, hardware, and/or other IT components in question.
- Manufacturer – the party that builds or develops the system or some of its components.
- Vendor – the party that sells and integrates the system (often the same as the manufacturer).
- Client – the party that purchases and uses the system.

The patterns shown aren't mutually exclusive; in some cases, two or even three apply, and in other cases it's a toss up as to which to select (for example, "Faulty Towers" v. "Never-ending Story"). More importantly, these few patterns can be used to describe virtually every case we found. They are presented below in roughly the order of frequency of occurrence within the survey results.

Finally, it's worth noting that these patterns *per se* are not unique to information technology; these same problems arise in other areas of commerce. IT lends its flavor in the inherent challenges and instabilities of IT development projects, the lack of professional standards and practices, and the common misunderstandings and lack of information concerning IT found from time to time in intelligent and educated business professionals.

PATTERN 1: FAULTY TOWERS

Summary: The client buys the system from the vendor. The client then claims that the system is defective, i.e., it has errors during operation, crashes, and so on. The vendor makes attempts to repair it, allegedly with limited and unsatisfactory success. In some cases, the client ends up returning the system and acquiring a new one from a different vendor.

Causes: Unfortunately, quality standards among IT systems and projects vary widely. For reasons outside of the scope of this paper, IT manufacturers often develop, vendors often sell, and clients often buy IT systems with quality problems that most people wouldn't accept in a minor household appliance, much less a software and/or hardware system costing hundreds, thousands, or millions of dollars. Likewise, the hidden and intricate nature of most IT systems—in particular, the ethereal nature of software—can hide defects and their root causes, even from IT professionals.

At the same time, there are cases where it appears that the client is claiming unacceptable defects in order to get out of a lease, contract, or other payment agreement. But what constitutes "acceptable" quality? No IT system, however well engineered, is perfect in all aspects, and the costs associated with increasing quality rise sharply after a certain level of quality is reached.

Recommendations: All parties to the IT project should agree ahead of time to specific expectations, promises, and contingencies regarding each of the areas of quality given above. For example, the system specifications should include not just the required functionality, but should also spell out any performance requirements or constraints, compatibility requirements, anticipated lifespan, and acceptable levels of defects.

[5] The term "pattern" has a specific usage in software engineering, describing a non-obvious solution to a problem in a given context; see **Design Patterns** by Gamma et al. (Addison-Wesley, 1995). The patterns in this study would actually be termed "anti-patterns" in software engineering circles, since they describe problems rather than solutions. For basic information on patterns and pointers to more information, see http://www.enteract.com/~bradapp/docs/patterns-intro.html.

Both parties should also clearly and unambiguously define key terms, conditions, and activities such as the meaning of "beta testing" or the standards for judging whether the client has accepted the system. In the IT world, accepting a system can occur at many different times, such as when it has passed a series of agreed-upon tests ("acceptance testing") and has been in operation for a certain period of time with no serious defects appearing. If all parties are not willing or able to do so, that's a strong warning sign that a dispute may well emerge. However, chances are the exercise of creating such a document will in itself flush out potential problem areas well in advance of any signing, payment, or delivery.

PATTERN 2: IRRATIONAL EXUBERANCE

Summary: The vendor makes claims for the functionality and/or performance benefits of the system. The client buys the system and has it installed. The client then believes that the system does not have the claimed benefits (performance and/or functionality). In some cases, the client ends up returning the system and acquiring a new one from a different vendor.

Causes: At times the vendor's IT sales representative doesn't have sufficient technical experience with the product in question and may make statements, promises, and assurances that lack grounding in reality, knowingly or unknowingly. And, of course, the sales representative has a strong vested interest in making the sale and therefore may exaggerate somewhat to close the deal.

It is telling that so many of the cases surveyed include charges of fraud.[6] However, even when these statements can be documented, they are dismissed in many cases by the judge as "statements of opinion" or "sales puffing." Likewise, words and phrases commonly used in the IT industry—such as "beta version", "performance", and "free of defects"—are viewed in many cases as being too vague or subjective without express, written definitions of the terms.

On the other hand, the clients often succumb to irrational exuberance themselves. They see the vended system as a "silver bullet" that can slay the ever-challenging IT problems faced.[7] They underestimate the difficulty in installing and adopting new IT systems. They change requirements on the fly, sometimes without notifying the vendor. Or they simply use general statements by the vendor as an excuse for their own inability to make the system operate the way they had hoped. In some situations, the client may suffer from "buyer's remorse" as the expense and challenges of the new system become apparent, and they may look for reasons to terminate the deal.

Recommendations: Miscommunication, both inadvertent and deliberate, has always been a large factor in IT systems failures. Some of the more common "irrational exuberance" issues can be avoided by making sure both sides agree upon a common, written set of definitions, specifications, and time tables with regards to the systems in question. As questions and issues arise, both sides can refer to and, if necessary, revise the document. This document should also go up and down the chain of command in both organizations as needed to make sure all relevant personnel understand what is promised and what is expected.

---

[6] The third most common claim, as seen in Appendix A.

[7] See "No silver bullet: essence and accidents of software engineering", Frederick P. Brooks, Jr., *Computer Mag*azine, April, 1987; reprinted in **The Mythical Man-Month** (20th Anniversary Edition), Frederick P. Brooks, Jr. (Addison-Wesley, 1995).

PATTERN 3: THREE'S A CROWD

Summary: This pattern actually lumps together two sub-patterns. In the first, the client purchases an IT system from the vendor by way of a leasing firm. The client is dissatisfied with the system and stops payment, whereupon the leasing firm sues the client. In the second sub-pattern, the client hires a consultant to recommend, select, or add value to system(s), vendor(s) and/or manufacturer(s). Problems occur in the development, installation, and/or use of the selected and possibly modified systems and the client blames the consultant, who may in turn blame the vendor/manufacturer. In both cases, someone other than the client and the vendor is being impacted by alleged problems with the system.

Causes: In a vendor/leasing firm/client triangle, the client usually signs a lease with a "hell or high water" clause, obligating it for the full lease regardless of the quality and usefulness of the system. At that point, the client is stuck with that obligation, and in the cases reviewed, the court consistently upheld that clause.

The various vendor/consultant/client triangles typically boil down to mutual finger pointing, with each party seeking someone else to blame. Consultants were most at risk in cases where they were making recommendations to clients; some courts found that the consultants were acting in a professional capacity and thus were held to a higher standard.

Recommendations: In at least one leasing case, while the court upheld the "hell or high water" clause on behalf of the leasing company, it ultimately ruled against the vendor because the vendor agreed to assume full liability under the clause should the system prove unacceptable to the client. Clients should keep this in mind, recognizing that otherwise the vendor has only limited exposure, if any, should the system prove unsatisfactory.

As for consultants, value-added retailers ("VARs"), and other third parties, the need for clear communication and agreement among all parties is critical. Make sure that the client knows exactly what you are (and are not) providing; likewise, be sure you have confidence in whatever systems you are using, acquiring, or recommending.

PATTERN 4: THE NEVER-ENDING STORY

Summary: The client contracts with the manufacturer to develop and install a system. The project starts. The completion date slips. It keeps slipping. Each time the adjusted delivery date approaches, the project slips yet again. At some point, one of three things happens: the manufacturer/vendor abandons the project; the client cancels the project; or the manufacturer delivers a system that the client terms wholly inadequate and unacceptable. In some cases, the effort has gone on for years, with millions of dollars spent and little to show for it.

Causes: This is actually one of the most well-known patterns in software development, particularly for large, in-house projects. Entire books are devoted to this subject[8], so it's hard to summarize the causes here. Two factors show up time and again, though. One is a lack of clear, stable, and constrained requirements on the part of the client. The other is a lack of qualified technical managers and developers on the part of the manufacturer. Chances are that both factors occur in a given project failure, giving each side plenty of reason to point fingers at the other.

[8] For example, **Software Runaways**, Robert Glass (Prentice Hall, 1997) and **Software Failure: Management Failure**, Stephen Flowers (John Wiley & Sons, 1996); many other software engineering texts touch upon this problem as well.

Recommendations: To use one software engineering maxim, "Start out stupid and work up from there." Most failures of this type come from attempts to implement large, complex systems from scratch. Experience has shown that success in building such systems comes more often from implementing small, comprehensible systems that work, then evolving them into the desired large systems.

Beyond that, you should do a thorough risk assessment of the entire project at the start and take steps to reduce any risks and protect your interests accordingly. Again, this may sound obvious, but many system project failures, large and small, have come about because no one with sufficient authority was willing to raise risk issues at the start. Risk issues that should be addressed include the scope and inherent feasibility of the project, the stability of the client requirements, the development and quality practices of the manufacturer, and how realistic the estimates are for time, money, and other resources allocated.

PATTERN 5: UNPLANNED OBSOLESCENCE

Summary: The client buys a system from the vendor. Some time later, the client discovers that the system either no longer meet its needs or that the vendor/manufacturer will no longer support it.

Causes: Unplanned obsolescence cases are not the result of a natural flow of upgrades and replacements in IT systems. These cases stem either from a sudden abandonment of a product version or line by the manufacturer/vendor or from some built-in flaw and previously unknown (at least to the client) flaw, such as Y2K. The former usually stems from financial issues, with the vendor/manufacturer abandoning an unprofitable line; the latter from software engineering flaws.

Recommendations: Most vendors and manufacturers give sufficient advance notice of such phase-outs, with a migration path for current users and sometimes a support plan (usually expensive) for those clients who wish for whatever reason to continue to use the old version. However, market and other considerations can sometimes constrain such notification.[9] Still, any vendor or manufacturer planning an abrupt retirement of a product or product line had best provide a migration path for customers or be prepared to face exactly these type of lawsuits.

Clients should have contingency plans for migrating off IT technologies they use but do not control. The level of detail should correspond to the size and stability of the company in question, the proprietary nature of the system technology, and its market share. Clients will worry less about products from internationally prominent manufacturers of mainframes, servers, workstations, PCs, and corresponding software. However, the smaller the firm and the more custom or proprietary the software, the greater the risk.

PATTERN 6: UNINTENDED CONSEQUENCES

Summary: The manufacturer makes some change in the functionality or configuration of the system, which is already in use. The change results in unpleasant or unintended consequences for one or more clients.

[9] For example, avoiding the "Osborne effect", so named for Osborne Computer Company ("OCC"), an early 1980s manufacturer of portable computers. OCC--with just one model on the market--put itself out of business largely by announcing a new and significantly improved model before that new model was ready to ship. OCC's cash flow dropped to almost zero as customers stopped buying the existing OCC model in anticipation of the new one, and the company—already under great financial pressure—went bankrupt before it could get the new model out.

Causes: Someone at the vendor/manufacturer mandates or proposes a change, and it gets made without careful consideration of its impact on existing systems or proper testing. The resulting consequences for one or more clients lead to legal action. Note that the only class-action lawsuits represented in our survey all fall into this category and all involve commercial products or services.

Recommendations: Think carefully about the consequence of changes. Test modified products thoroughly. Roll them out in limited numbers with trusted, friendly clients to flush out problems. Act quickly to fix problems that show up.

LESSONS TO LEARN

Having reviewed these cases and the patterns they exhibit, some practical suggestions come to mind.

LESSON 1: GET EXPERT LEGAL AND IT GUIDANCE BEFORE SIGNING ANYTHING.

Most of the legal pitfalls in IT business deals are well documented. Too often though, clients, vendors, and manufacturers sign contracts and agreements without having them reviewed by lawyers who understand IT-specific pitfalls. This may seem obvious, yet case after case in the survey focuses on the terms of signed agreements and the efforts by parties on one or both sides to find promises and protections that the courts find were never put in writing.

It's also a good idea to run the agreement past technical people in relevant IT departments. For vendors and manufacturers, such people are likely to point out "overly enthusiastic" promises that the vendor might have difficulty keeping. For clients, such people can help spot flaws in technical commitments that relate to what is to be delivered and when.

LESSON 2: SPECIFY CRITICAL TERMS AND ARTICULATE PROTECTIONS BEFORE AGREEING TO A SALE OF GOODS OR SERVICES.

In many of the cases we surveyed, one side or the other ended up losing because the parties failed to define important technical terms with sufficient specificity. When reaching agreement on what will be delivered and when, parties should endeavor to define clearly and in detail key terms. These include the various types of quality assurance (QA) that will be performed, the IT requirements that will need to be met for the client accepting the system, and the process for having the vendor/manufacturer deal with defects that show up after acceptance.

IT quality assurance aspects and activities that you could define and agree upon might include: expertise; guidelines and standards; metrics (measurements of progress); quality reviews; the various forms of testing; defect management; configuration management; and product release cycles, including technical support, maintenance, and upgrades.

Likewise, a good starting point for defining acceptance criteria is the list of quality attributes given earlier: reliability, performance, functionality, compatibility, lifespan, deployment, support, and cost. If both sides have a clear, written understanding in advance of what the system will and won't do in each of those areas, the project has a higher chance of success.

LESSON 3: ACT QUICKLY WHEN PROBLEMS ARISE.

The foundation of both the "Faulty Towers" and "Never-ending Story" patterns is the client's willingness to let the situation drag out for months or even years in spite of repeated failures to deliver or perform.

Why does this so often happen? Three factors, individually or combined, underlie this unwillingness to pull the plug on a project. First, the client may have a substantial monetary investment in the project and sees going forward as a better option than pulling out. Second, the executives and managers within the client firm who made the decision to acquire or develop the system in the first place often have a strong professional and personal interest in seeing it go forward and not having their original support for the project held against them. Third, migration off old systems that has already occurred may make going back difficult.

Even so, letting such projects go ahead as they are is almost always the worst decision. Case after case, both in our survey and in software engineering literature, shows that without direct and dramatic intervention, the client will end up spending more time and money in a project that in the end fails anyway. Instead, the client should document clearly the problems and attendant risks and consequences, then review them internally to determine the best course: proceed ahead, redirect the project (such as by scaling it back or changing the requirements), or cancel the project altogether. The client should then work with the vendor/manufacturer to achieve the desired goal.

LESSON 4: REMEMBER THAT NEW TECHNOLOGY ENTAILS RISKS.

The phrase "new technology" here refers to either relatively new commercial IT system offerings from a commercial vendor or manufacturer, or custom IT systems being built specifically for the client. While many IT system project failures stem from mismanagement or simple human failings, some founder on genuine technical issues: that which is being attempted may just not be feasible. Adele Goldberg, an IT software pioneer, has said, "Only optimists build complex systems." One might amend that to read "build or buy", but the corollary is that it is often optimists who end with project failures. Some healthy skepticism and caution, as well as a sober realization that large, complex IT projects have a high rate of late delivery or failure, should be taken into account when planning and negotiating the acquisition of such systems.

CONCLUSIONS

Most of the observations and recommendations made in this paper are common sense. That said, we have to ask ourselves why such common sense is so often set aside or ignored. The court documents and other literature on these cases don't spell it out, but professional experience suggests that it largely boils down to human failings, ranging from naïveté to dishonesty.

The patterns of litigation involving information technology are readily identified. Judicious thought, consultation, and agreement on detailed terms, especially before entering into a legal agreement, will go a long ways to avoiding those patterns, reducing the risk of or need for a lawsuit. And that, in the end, is better for all parties involved.

APPENDIX A: THE PRICEWATERHOUESCOOPERS IT SYSTEMS FAILURE LITIGATION DATABASE

The PricewaterhouseCoopers IT Systems failure Litigation database contains information on 120 legal disputes filed in the period 1976-2000.

The information we sought to gather for each case includes:

- Case name, case type, court and jurisdiction, judge assigned, case number, and date filed;
- Litigants and law firms representing each;
- Causes of action;
- Relief sought, relief awarded, and settlements (if any);
- Case history;
- Sources of information about the case.

For many cases we weren't able to gather all the information but did the best we could with the sources available. We also added a summary of each case as well as notes for different areas. Finally, we made a determination of the industries involved (where possible) and--after finishing our initial analysis--of the patterns and quality issues that appeared to apply.

SOME DATABASE STATISTICS

As we mentioned in the body of the paper, this survey was not an exhaustive one. It's not even clear that it could be considered a sufficiently random sample, since the basic list depended significantly upon what was deemed of interest by the editors of *Computer Law and Tax Review* (*CLTR*) from month-to-month and year-to-year. Because of these limitations, drawing general conclusions from any statistical analysis of the database carries risks.

With that caution, a few trends suggest themselves. First, the number of such cases has been on an upward trend over this period. Consider the following graph, showing the filings for each three-year period:

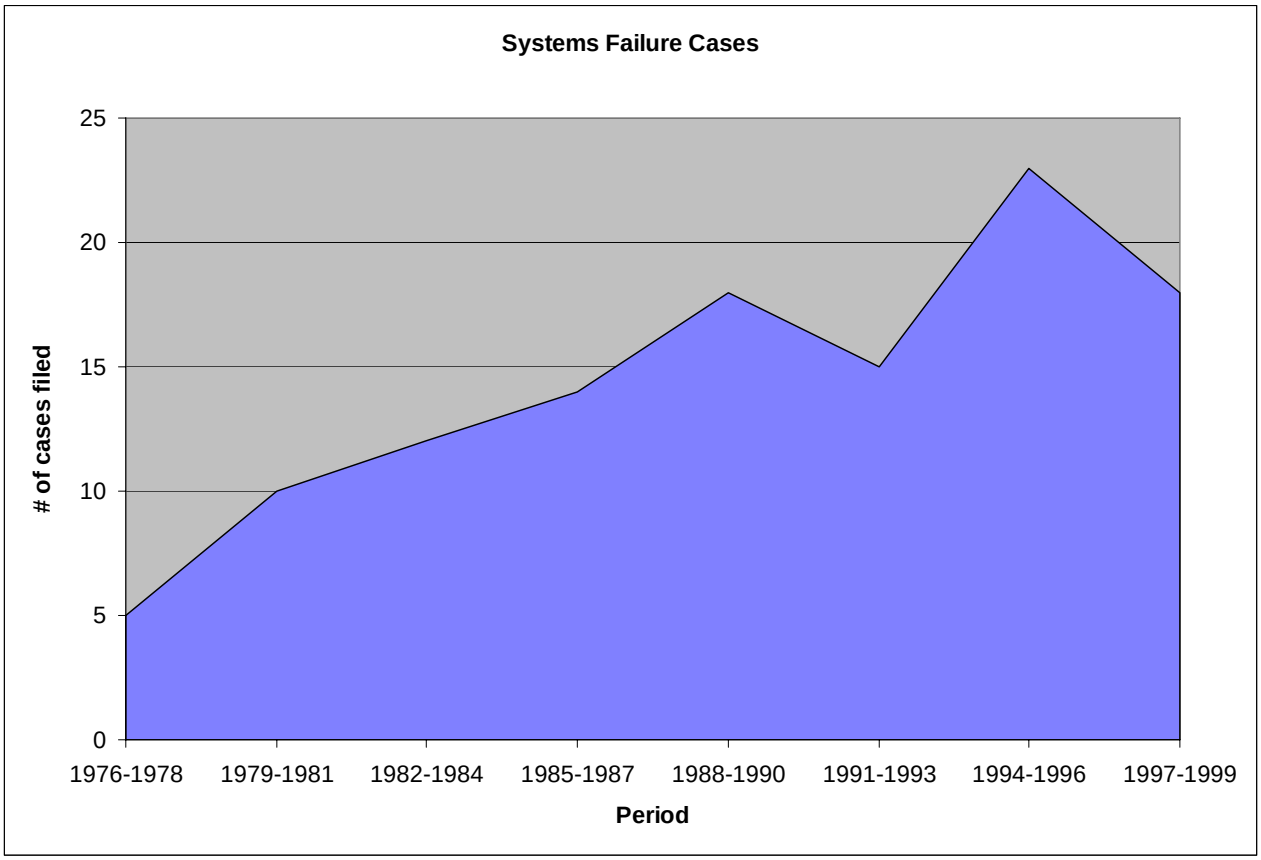

Chart 1: Date of filing for cases in the Systems failure database.

While on a year-to-year basis there's a lot of up and down movement in the graph, the overall trend is higher over time. The drop in 1997-1999 reflects a time lag in our ability to gather data on the most recent cases, rather than an actual downward trend in such legal actions.

The breakdown of industries involved yields no clear results other than the obvious and expected one: in most cases, one of the parties is an information technology company of some sort. This includes firms selling or producing computer software, hardware, services, or complete systems. The second most common industry was manufacturing, with leasing close behind.

Industry - Plaintiff	Total
Accounting	3
Advertising	1
Apparel	2
Automotive	3
Chemical	1
Class Action	3
Computer Hardware	3
Computer Services	1
Computer Software	5
Computer Systems	9
Construction	2
Consulting	3
Consumer Goods	1
Distribution	1
Education	2
Entertainment	2
Finance	1
Food	1
Food Retail	2
Food Wholesale	1
Government	1
Graphic Arts	1
Health Care	5
Holding Company	1
Hospitality	1
Individual	4
Insurance	6
International Distribution	1
Leasing	10
Legal	2
Manufacturing	11
Petrochemical	1
Publishing	1
Real Estate	2
Retail	2
Services	2
Shipping	1
Small Business	1
Telecommunications	2
Transportation	2
Unknown	11
VAR	3
Wholesaler	2
Grand Total	120

Industry – Defendant	Total
Agriculture	1
Automotive	1
Computer Design	1
Computer Hardware	4
Computer Services	6
Computer Software	14
Computer Systems	61
Construction	1
Consulting	3
Entertainment	1
Government	3
Graphic Arts	1
Health Care	6
Individual	1
Machine Tools	1
Manufacturing	3
On-line Services	1
Petroleum	1
Real Estate	1
Retail	2
Semiconductor	1
Supplier	1
Telecommunications	2
Unknown	3
Grand Total	120

Table 1: Industries as plaintiffs or defendants for cases in Systems Failure database.

We already presented the relative order of frequency of the six patterns, but Chart 2 shows the actual numbers. Having read through the rulings, court filings, published articles, and other sources on these cases to determine what patterns exist, we then used those same documents to assign patterns to each case. Often, a given case reflected two or even three patterns. In such cases, we made our best determination as to which pattern predominated and which were secondary. Chart 2 reflects that as well, showing for each pattern how often it was the principal pattern and how often it was a secondary pattern to one of the others. Also, several cases didn't fit any of the patterns or we lacked sufficient information to determine what the pattern might be.

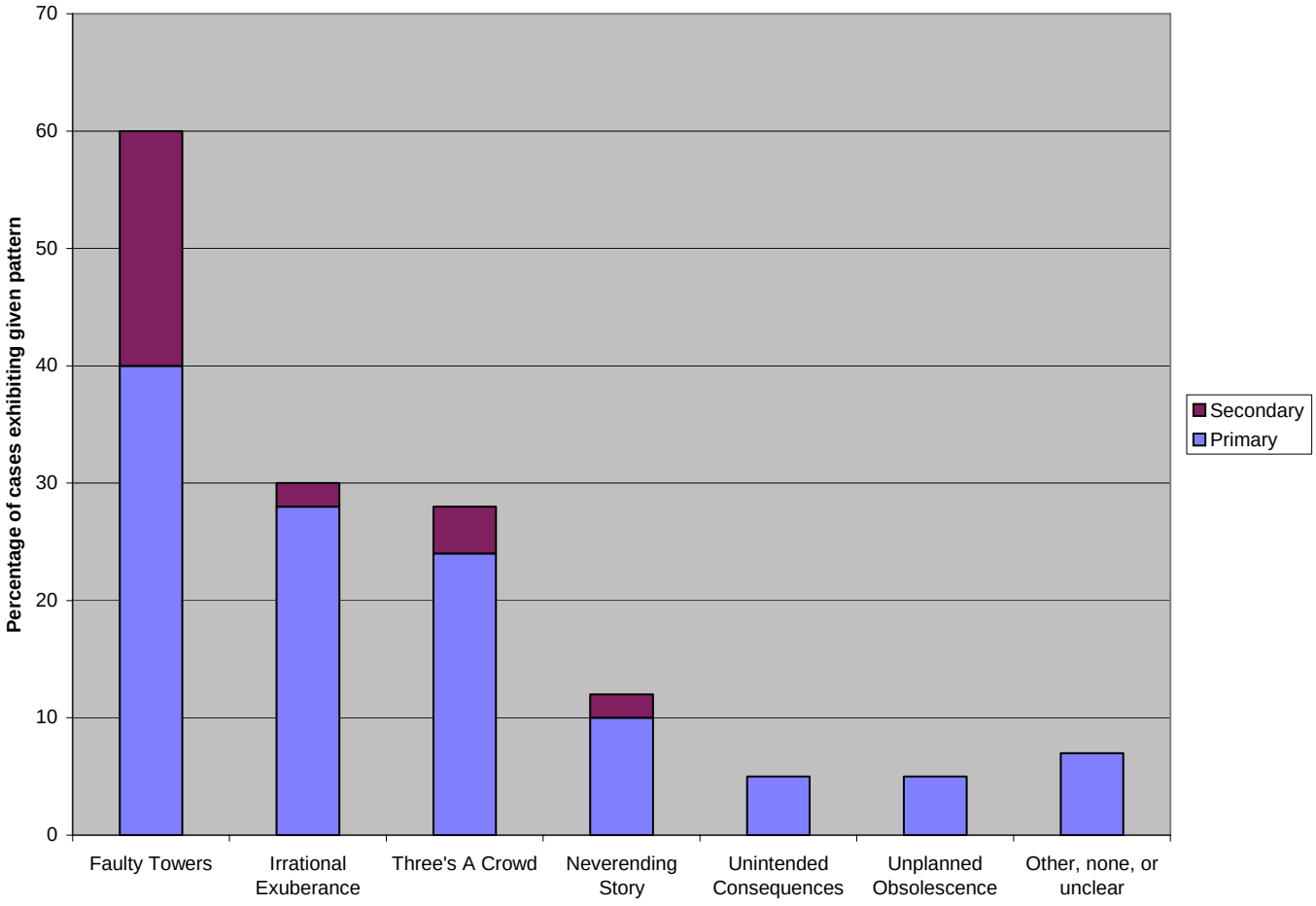

Chart 2: Frequency of patterns among cases in IT Systems Failure Database

The majority of cases in our database have two to four claims alleged by the plaintiff. Chart 3 shows the relative proportion of the dominant types of claims made in such cases:

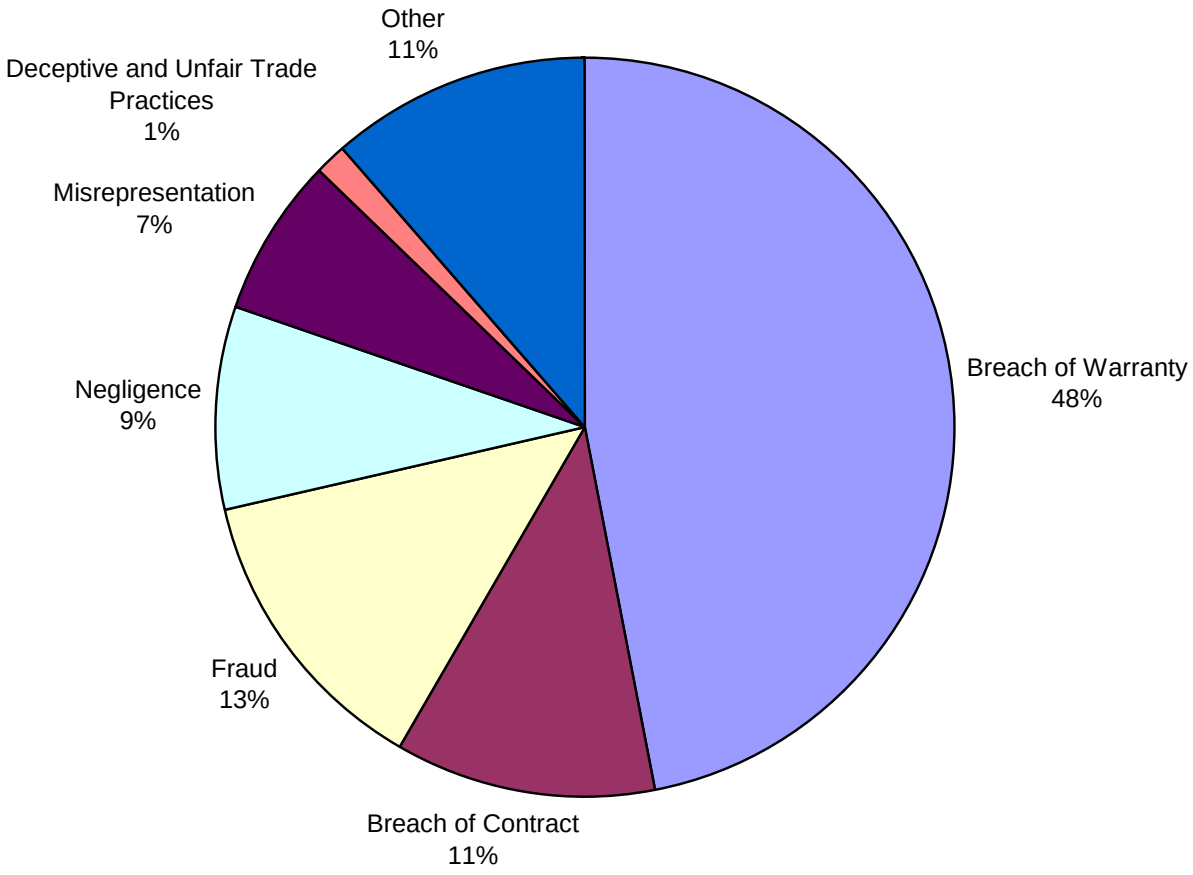

Chart 3: Claims made in cases in IT Systems Failure database.

The dominant claims are breach of warranty, breach of contract, fraud, negligence, and misrepresentation. This is not surprising, given the nature of the cases. What is telling is how often the court dismissed claims of breach of warranty because of explicit--and often boilerplate--exclusion in the contract or licensing agreement.

APPENDIX B: SELECTED WORKS ON SOFTWARE ENGINEERING

This list could comprise dozens of titles, all worthy of consideration (though many, unfortunately, out of print). The selected titles below are presented because they focus on how to help IT projects succeed and--more importantly--why they so often go wrong. Also, they are all currently in print, plus most are relatively thin and can be understood by non-technical readers.

The Mythical Man-Month: Essays on Software Engineering (20th Anniversary Ed.), Frederick P. Brooks, Jr. (Addison-Wesley, 1995).

201 Principles of Software Development, Alan M. Davis (IEEE Computer Society, 1995).

Peopleware: Productive Projects and Teams (2nd ed.), Tom De Marco and Timothy Lister (Dorset House Publishing, 1999).

Software Failure: Management Failure, Steven Flowers (John Wiley & Sons, 1996).

Principles of Software Engineering Management, Tom Gilb (Addison-Wesley, 1988).

Software Runaways, Robert L. Glass (Prentice Hall, 1998).

Assessment and Control of Software Risks, Capers Jones (Yourdon Press, 1994).

Debugging the Development Process, Steve Maguire (Microsoft Press, 1994).

Software Project Survival Guide, Steve McConnell (Microsoft Press, 1998).

Productivity Sand Traps and Tar Pits, Mike Walsh (Dorset House Publishing, 1991).

The Psychology of Computer Programming: Silver Anniversary Edition, Gerald M. Weinberg (Dorset House Publishing, 1998).

Death March*,* Edward Yourdon, (Prentice Hall, 1997).

APPENDIX C: ABOUT THE AUTHOR

Bruce F. Webster is an internationally recognized expert on information technology (IT). He has been an invited speaker at international IT conferences in Russia, Japan, Central America, and the Middle East. At the invitation of the US government, he has given on a regular basis private IT-related briefings to representatives of other nations. He has given presentations at private conferences of the World Bank, the US intelligence community, and U.S. Congressional staff. He has testified about information technology issues three separate times before Congress and has provided analysis and documents to Senate and House committees. Likewise, he has been an invited speaker and participant at public and private conferences at the Center for Strategic and International Studies.

Webster has spoken at numerous US-based conferences, including: Mealey's IT Systems Failure Conference (originator and co-chair); ABA Section on Litigation Winter Leadership Conference; Mealey's Internet Law Conference; the Blue Cross/Blue Shield Millennium Solutions Conference; the National Association of Securities Dealers Conference; the University of Chicago Graduate School of Business Annual Management Conference; the MBA National Technology in Mortgage Banking Conference; and several of the Software Development and ObjectWorld Conferences.

Webster personally has been involved in developing, delivering, analyzing, and advising on information technology for over 30 years in a broad range of fields: e-commerce, finance, communications, commercial software, manufacturing, aerospace, and law. He has done consulting with or at a variety of companies during the past fifteen years, including many Fortune 100 firms and major information technology companies. He also taught computer science for two years (1985-1987) at Brigham Young University.

Webster also spent more than a decade in the commercial software market, helping to start up two different software companies and raising over $7M in venture funding for one of them. He has contributed to over a dozen commercial software products on half a dozen platforms, acting as chief architect twice. He has served as chief technical officer for two different companies.

Since 1980, Webster has written and published over 150 articles on computer industry analysis, product evaluation, and software development. Since 1989, he has authored four popular and well-received books on information technology issues while contributing to two others.

Webster is a Principal at Bruce F. Webster & Associates LLC. He provides expert analysis in matters involving information technology, with a focus on system development, project failure, intellectual property, web and internet technologies, and software engineering. He has worked with several large corporations, providing advice on enterprise systems architecture, software technology, development methodology, and intellectual property practices. He has also consulted in several dozen legal cases, both in the United States and Japan, and has qualified and testified as an IT expert in court, in arbitration, and in deposition. He is a graduate of Brigham Young University (BSCS, 1978) and did graduate work in computer science at the University of Houston/Clear Lake City. He is a member of ACM (Senior Member), IEEE, ISACA, and the American Bar Association (Associate Member). He and his wife Sandra live just outside of Denver, Colorado.

Webster can be reached at (720) 895-1405, at bwebster@bfwa.com, or via http://bfwa.com.