

Getting over Y2K

Computer rough spots: Litigate, settle or ignore?

By RONALD I. RAETHER Jr.

Raether is a associate at Faruki Gilliam & Ireland in Dayton, Ohio.

As I watched a computer expert connect various clouds, squares and circles, I realized that no one had written about how technical and business considerations affect Y2K litigation. As a result, I began to write this article not knowing whether Y2K would affect other companies as it already had one of my clients involved in a Y2K-related class action, where I am counsel. As the New Year came and went, without many Y2K calamities, I felt like the Cleveland woman who had purchased a four-year supply of Spam. What should I do with this article?

With the year 2000 now here, many believe the lack of mayhem at the turn of the millennium means the unlikelihood of Year 2000-related lawsuits. That may not be. The Gartner Group estimated that less than 10 percent of problems would occur in the first two weeks, while the remainder would hit during the rest of the year. Computer problems generally must be diagnosed after some failure has occurred. These failures could relate to monthly, quarterly or year-end activities, which may explain, in part, why not all Y2K disruptions appeared immediately following the New Year. Also, the failure may be discovered long after the computer problem began corrupting data.

However, even in the first two weeks, Y2K disrupted business - such as small businesses using Cybercash's credit card payment system, which following the New Year billed customers day after day for the same transaction. ("Ashes of the Year 2000 Problem," *New York Times*, Feb. 28, 2000.) Unfortunately, your company may still face or anticipate bringing Y2K-related lawsuits. Even without Y2K, companies that use technology (which means virtually everyone) may face computer-performance-related problems similar to Y2K.

This article reviews two important Y2K-related cases and their relevance beyond Y2K for businesses that use or sell computer products. The analysis, however, should not end with these cases. Software performance litigation is rarely "off the shelf." Instead, this article will also discuss several technical and business considerations that, whether

Getting

over
Y2K

Computer rough spots: Litigate, settle or ignore?

By RONALD I. RAETHER Jr.

As I watched a computer expert connect various clouds, squares and circles, I realized that no one had written about how technical and business considerations affect Y2K litigation. As a result, I began to write this article not knowing whether Y2K would affect other companies as it already had one of my clients involved in a Y2K-related class action, where I am counsel. As the New Year came and went, without many Y2K

Raether is a associate at Faruki Gilliam & Ireland in Dayton, Ohio.

Thomas Tomak

Business Law
Today

#8

September/October 2000

dealing with off-the-shelf software or highly sophisticated customized turn-key solutions, are generally important to any business evaluating potential or existing computer-performance issues.

The majority of Y2K performance-related cases filed to date have concerned whether a software supplier should be compelled to provide, free of charge, a Y2K-qualified system. The claims arise in contract and tort, including breach of contract, anticipatory breach, fraud, violation of state consumer protection laws and breach of the covenant of good faith and fair dealings - the same causes of action are present in most software performance disputes. The one common element in these cases is the court's refusal to impose an obligation to provide a free Y2K upgrade without express language in the contract - but what impact do these decisions have outside Y2K? A review of the cases is the best place to start.

In *Paragon Networks Int'l v. Macola Inc.* (Case No. 98-CV-0119, Court of Common Pleas, Marion County, Ohio), plaintiff Paragon Networks International purchased shrink-wrapped accounting software, developed by defendant Macola Inc. from a third-

party retailer. Inside the package was a licensing agreement that expressly excluded all express and implied warranties, stating that the product was being sold “as is, without warranty of any kind.” On the outside of the package were similar terms.

Macola’s software could not process four-digit year dates. As a result, Paragon filed a complaint for breach of contract and fraud, and sought class-action status. Paragon alleged that Macola’s advertising, including its Web site, promised a solution “You’ll never outgrow.” Paragon based its fraud and contract claims on its belief that this statement was fraudulent or created an express warranty. Macola filed a motion to dismiss, which the trial court sustained.

The court of appeals affirmed, relying on the express language of the contract’s integration clause and limited warranty. *Paragon Networks Int’l v. Macola Inc.* (No. 9-99-2, slip op. at 7-9; Ohio Ct. App. Marion Cty. Apr. 28, 1999.) The court of ap-

peals upheld the dismissal of the warranty claim because the ads and “other marketing puffery” could not create an express warranty that contradicted the express terms of the contract’s limited warranty. *Id.* at 8-9.

The court of appeals affirmed dismissal of plaintiff’s fraud claim because “[p]arties may not . . . prove fraud by claiming that the inducement to enter into an agreement was a promise that was within the scope of the integrated agreement but was ultimately not included in it.” *Id.* at 10. That court



reasoned that plaintiff’s allegations regarding manufacturer misrepresentations about the useful life of the software implied a warranty, which was in direct conflict with the contract’s express limitation of warranty. *Id.* at 11.

Similarly, in *ASE Limited v. INCO Alloys International Inc.* (AA55-199-0127-98-DEV), an arbitrator relied on a contract to reject Y2K-related claims. The plaintiff, INCO Alloys International Inc. (INCO), hired ASE Limited (ASE) to act as project manager during the development of software, which

Recent decisions can have impact outside Y2K.

was to be INCO's Inventory Management and Control System (IMCS). Award of Arbitrator, p. 1. The contract, executed on Sept. 1, 1995, was for the delivery of software at various stages over five years.

Following a 12-day hearing, the arbitrator, William M. Wycoff, denied INCO's request for Year 2000 remediation. After rejecting arguments that ASE had breached the contract for other reasons, the arbitrator addressed INCO's claim for damages. *Id.*, 14. The arbitrator denied the claim because (1) "Year 2000 remediation is not set forth as an engineering field in the contract"; (2) there was no evidence that the parties "ever added [Year 2000 mitigation] to the contract by a writing signed and agreed to by both parties"; and (3) a June 16, 1996 blueprint stated "Year 2000 mitigation (not included in the IMCS Scope)." *Id.* Absent an express agreement, failure to provide Year 2000 remediation was not "a material default in the contract by ASE." *Id.*

Both decisions are consistent with well-established jurisprudence: Unambiguous contract provisions should define the vendor's obligations and user's rights. Absent express language to the contrary, the vendor should not be the insurer of its product - warranties should be limited to the duration and scope set forth in the contract. This conclusion is consistent with the realities of software development - no software program can be 100 percent tested and delivered without any "bugs" and still be affordable.

The same is true for complicated customized solutions, as demonstrated by ASE, where both parties were sophisticated merchants. The contract demonstrated both INCO's knowledge of Y2K and the absence of any affirmative obligation to address Y2K. Again, ASE is consistent with the realities of software programming. In developing software, anything is possible with sufficient time and money. In the end, for custom solutions, the end user decides whether the benefit justifies the investment.

Nevertheless, the consumer is entitled to software that performs as promised or as may be implied from industry norms and conventions. For off-the-shelf software, like Macola's accounting solution, the application should provide the basic functionality described in the sales literature. For more sophisticated products, the client should get what was promised, limited only by the realities of software

programming. How long the software will meet the needs of the consumer or whether a program should work under all conditions (expected or unexpected),

however, is a complicated question that requires analysis of the facts of each situation.

As a result, the analysis should not end with the contract. Instead, technical and business considerations also are important if the contract is ambiguous or to determine if fraud or some other tort is actionable. These issues include the:

- complexity of the system;
- sophistication of the parties;
- access to the source code;
- dependence on third-party suppliers;
- useful life of the product; and
- economies of the market.

While not exhaustive, this list provides the starting point to analyze what a customer could reasonably expect.

Development realities affect what reasonably can be accomplished given limits in technology - often called bandwidth. Unfortunately, many people mislabel a system's inability to perform a function, such as handling four-digit date code, as "a defect." Programming decisions necessitated by development or market limits are not defects. Unlike a poorly designed or manufactured product, where the consequence in most circumstances is unknown to the designer, using a two-digit date code, in the example of Y2K, was in most instances a conscious decision caused by necessity.

Until only recently, even a small amount of memory was expensive. Using a four-digit date code increased significantly the amount of memory necessary to process and store date information. This fact imposed two limits on the marketability and usefulness of software: increased total cost of a system; and reduced memory available for desired functionality.

Thus, even if the contract is ambiguous, for solutions developed when memory was expensive, using a two-digit date code was not a design defect, but a necessity. Moreover, use of the two-digit date code did not prevent the system from operating as designed and as the end user expected, that is, operating properly at the time purchased. Absent

Should industry norms control the interpretation?

representations to the contrary, the end-user received what could reasonably be expected. The same is true for other development techniques required by technical realities, such as hard-coded data, workarounds or simply leaving out functionality not justified by the cost.

However, the inverse is true as well - consumers should get the best use of technology they are willing to purchase. Many solutions are developed over old code - designed before certain advances. The new solution has the function desired by the consumer, but is limited by the old development techniques used in the older code. The difficult question is - should the industry norms at the time the software was developed or sold control the interpretation of ambiguous contract language?

The answer, of course, depends on the:

- complexity of the system;
- purchase price;
- industry;
- competition;
- needs and size of the customer base; and
- most important, any oral representations.

Just as we understand that used cars have certain risks, we should appreciate the limits of old code and technology. The interpretation of a contract, or the existence of fraud, depends on the consumer's knowledge of the product and its design or intended use.

Moreover, technology today is rapidly advancing. As many of us have experienced with our personal computers, improvements and significant changes occur frequently, such as Windows 3.1., Windows 95, Windows 98, Windows 2000. With each of these improvements, another question arises - how long can someone reasonably expect to use any software or hardware? Reasonableness should not be determined based on extremes. While the most sophisticated users do not create the standard, neither does the recalcitrant user who refuses to make capital investments to remain competitive.

Software and hardware come in many different forms and serve different markets and functions. These differences must be understood when assessing whether design techniques used, such as the two-digit date code, were reasonable and consistent with user expectancy. Complex systems require more memory; using two-digit date codes frees memory

for desired functions. Complex systems often require customization where the end user is often heavily involved. Thus, a customer involved in the decision to

use a two-digit date code or eliminate some other function has little basis for complaint.

The same is true for consumers who purchase off-the-shelf software where no future enhancements or upgrades have been expressly promised. In addition to the obvious business arguments - you get what you pay for - technical considerations justify the refusal to impose burdensome development obligations.

Alteration of any program depends on access to the source code - the work product of the programmer. It is the source code that must be analyzed and changed to permit a program to handle four-digit dates, add a function or make other changes. Depending on the history of the program, the source code may be unavailable to the current owner. Acquisitions, mergers and the purchase of products or subproducts extend the chain of ownership and increase the likelihood of the source code being unavailable.

Delivery of the source code also could be intentionally excluded from the license or sale of software. Where the selling party intends to continue competing in the market, it may not want the new owner to have the ability to make improvements or otherwise alter the program. While reverse engineering makes it possible to change programs without the source code, this process is very expensive, often not feasible and may violate copyright laws. To impose an obligation to make improvements under these conditions is not only unjust, but may be financially or legally impossible.

These complexities are compounded by the reality of how many businesses use IT today. Even if one purchases all computer needs from one source, it is unlikely that the vendor developed or has control over every element of the software, which is one reason to clearly define responsibilities in order to avoid the inevitable finger pointing when problems arise. Many systems have been developed by multiple parties or depend on the operation of other programs or hardware, which affects the analysis in two ways.

First, from the operating system - DOS, Win-

dows, Unix - to the Basic Input/Output System (BIOS), software is written to depend on data and platforms provided by outside resources. If dependent on third-party software, such as receiving two-digit date codes, then changing the software will not necessarily allow the product to operate as intended. Second, most license agreements permit access to source codes only in limited circumstances - namely bankruptcy or insolvency. As discussed above, without access to source code or cooperation of the third party, necessary changes cannot be made.

The motivation of the consumer and vendor may further affect this analysis. It is important to consider the specifics of the market, such as:

- sophistication of the customer;
- technological dependency of consumers;
- competition with other vendors;
- expected return on investments; and
- technological advances.

For most business applications, why a system is purchased is answered easily - reduced costs through enhanced efficiencies, which should result in increased profits. The solutions should increase productivity and provide information and data to improve marketing and increase profits. For larger, more sophisticated businesses, being competitive requires use of the latest innovations and adaptations of technology - direct mailings, improved customer service, customer profiles - all now made convenient and cost effective by computers.

For vendors, the customers' needs and desires control. Vendors compete to acquire market share by providing new functionality to serve the needs of their customers. The more sophisticated the client base and product, the more creative and innovative a vendor must be to survive. Vendors must continue to enhance solutions, develop new systems and discontinue old systems to maintain revenue to support research and development to keep their businesses viable. Continued support of an old system with a declining customer base may mean loss of market share and eventually insolvency. Vendors must be assured that these research costs can be recovered by sales or related service fees. Absent this potential, vendors can't survive.

In addition, labor resources are limited. The lack of programmers and computer specialists may

The analysis begins with the contract.

affect the vendor's ability to meet the customers needs.

The Year 2000 issue is a worthy example. If the useful life of a product is

limited, then expending resources on a product soon to be replaced makes little economic sense. Market economies support discontinuance of a product's use and affect whether a party should reasonably expect a Y2K fix.

What does all this mean? As the cases demonstrate, the analysis begins with the contract. Whether the warranty is limited or extends into the future may determine whether a free fix is required. Breach-of-contract claims also begin with the express language of the contract. However, the analysis may require a review of the facts relative to the vendor, the consumer and the product.

Macola provides a good example. If *Macola's* accounting application lacked an Internet banking and reconciliation function, should *Macola* be required to provide this function free of charge? The answer, of course, depends on the circumstances. Where ambiguity exists, the market realities discussed in this article may apply. Thus, in the above example, if the consumer knew the product was developed before the Internet technology was available, that the system was based on old code, that the function depended on unrelated third parties, or some other mitigating factor, then the vendor should not be responsible. Changing these facts would affect the vendor's liability.

To navigate these complexities, I suggest discussing the product with a member of your IT department or hiring an outside expert and addressing the issues raised above. Certainly, as with any case, a thorough review of the documents as well as interviewing the critical witnesses should occur before deciding on a course of action.

In the end, the consumer's rights will be based on the contract language and what reasonably could be expected under the circumstances. The realities of software development require as much. The law must balance the value of innovative and affordable software with unconditional guarantees. At a minimum, courts will probably continue to interpret contracts consistent with established rules of contract construction and market realities. ■