The Journal of Quality Assurance Institute, Vol. 13, No. 4, October 1999, pp. 14-20, 24-26

Beyond the Y2K Compliance: A System Approach to Solving Year-Digit and Leap-Year Problems

Eldon Y. Li

College of Business, California Polytechnic State University
San Luis Obispo, CA 93407, U.S.A.

Abstract

The Y2K crisis around the world can be attributed to the simple problems of year-digit and leap-year calculations in computer programs. This paper identifies the sources of the Y2K problems and proposes remedies to these problems based on the system approach. The proposed solution process is applicable to all computer platforms experiencing the same Y2K problem. It is useful in combating any similar problem beyond the year of 2000.

Keywords: Y2K, year digit, leap year, system, network, input, output, process.

1. Introduction

The year 2000 is right around the corner. There is not much time left for us to resolve the Y2K problem. The Y2K problem is a simple leap-year, year-digit calculation problem. Although this problem is very simple to fix, it is very expensive because it involves a vast amount of code written since the beginning of computers. For example, the office of CalPERS reported reviewing over 2.3 million lines of mainframe computer code to identify potential Y2K problems [6]. To fix this problem, General Motors is expected to spend over $550 million while American Airlines may spend over $250 million [2]. Even a mid-size school like California Polytechnic State University will spend about $740,000 to bring its campus computer systems up to compliance [4]. According to the Gartner Group, this problem may cost $600 billion worldwide to fix. Ultimately, it may cost $2 trillion, right behind the $4.2 trillion cost of World War II [1].

Since the early 1990s large private corporations in the U.S. have begun their effort in resolving Y2K problems. Several industry-wide initiatives were formed. The Automotive Industry Action Group (see http://www.aiag.org) has been offering testing advice and maintaining a database of testing results for manufacturers and parts suppliers. The National Retail Federation (see http://www.nrf.com) has been sponsoring Y2K compliance testing and certification for electronic data interchange system among retailers and suppliers. The Telco Year 2000 Forum (see http://www.telcoyear2000.org), an organization of 8 major carriers, has been working with Bellcore to test the national's telephone networks. The IEEE (see http://www.ieeeusa.org) has drafted a standard, IEEE 2000.2, to help engineering companies evaluate and test source code for Y2K compliance. The Securities Industry Association (http://www.sia.com) has coordinated a series of tests under various trading scenarios and using susceptible trade dates to ensure the safety of its computerized security trading system. The American Bankers Association (see http://www.aba.com/aba/prod_serv.html) has introduced two detailed interactive manuals specifically designed for Y2K project teams, namely, "*Year 2000 Project Management Manual*" and "*Year 2000 Contingency Planning Manual*."

Massive Y2K tests have been taking place since 1998 because potential system crashes will begin to occur in 1999. Many legacy computer programs use "99" or "9999" to designate "end of file," "loop exit," or "missing value." The output produced by these programs in 1999 will be incorrect. Furthermore, many systems looking forward one year into year 2000 and will begin to fail or produce spurious results. According to *Software Magazine*, by now most large government agencies and private corporations should have addressed the Y2K issue. Nevertheless, smaller government agencies and non-profit organizations could still use some help in combating their Y2K problem [2]. The purpose of this paper is to describe the Y2K problem, to identify the sources of the problems, and to propose a system approach to tackling the problems. This may prove to be useful to those companies who are not yet Y2K compliant in managing their Y2K effort.

2. The Y2K Problem

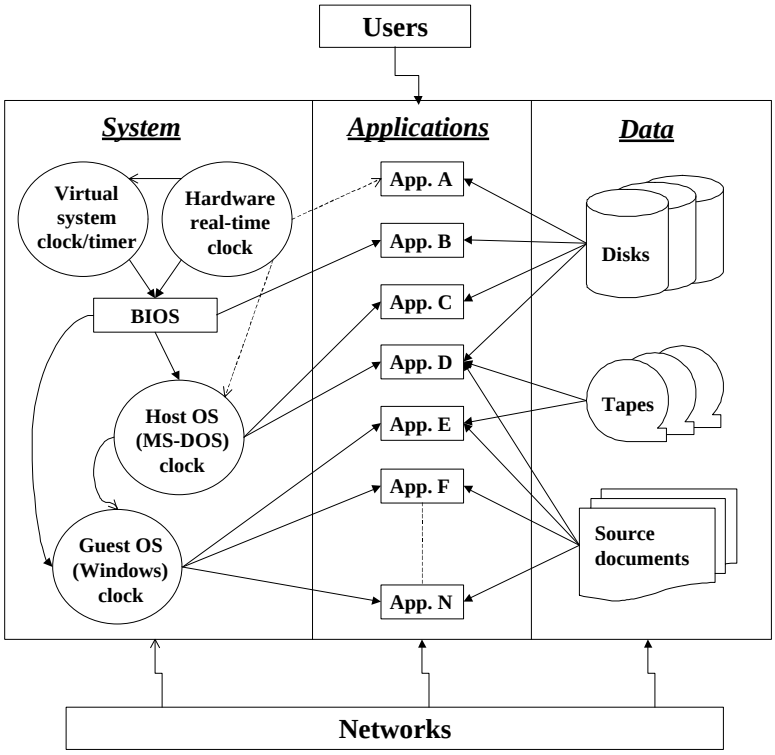
The Y2K compliance standard consists of two requirements. To meet these requirements, a computer system must be free from the following two problems:

1. ***The leap-year calculation problem:*** It is common for a programmer to divide the year value by 4 to determine if it is a leap year or not. The year of 1996 was a leap year, thus the year of 2000 must be a leap year. However, the year of 1900 was not a leap year although it is divisible by 4. That is, a simple division by 4 cannot accurately determine if a century year is a leap year. It is obvious that a century year not divisible by 4 is definitely not a leap year, but those divisible by 4 are not necessarily leap years as well. The necessary condition for a century year to be a leap year is that it must be divisible evenly by 400. Therefore, the year of 2400 is a leap year but the year of 2100 is not.
2. ***The year-digit calculation problem:*** This is a problem where the century digits are omitted from the year value by the computer system, giving 1900 and 2000 the same year value of "00." When a program code calculate the elapse year between two years across two centuries, a negative value is produced. For example, the elapse year between 1996 and 1990 is 6 years, but that between 2000 and 1996 is minus 96 years (i.e., "00" minus "96"). Furthermore, when comparing year 2000 with year 1996, a relational error occurs because "00" is less than "96." This error will make the program take a wrong course of action and produce a faulty output.

The leap-year calculation problem attributes primarily to the missing requirement of divisibility by 400. This problem may not need to be fixed because most business transactions will not use the year earlier than 1901 or beyond 2099. If our source code calculates its own leap years, there is no problem at all as long as the year value is within this period. On the other hand, if our source code relies on the Date function provided by the system software. This safety period is restricted by the system's real-time clock (RTC). On the IBM PC/AT or later models, the RTC has an allowable period between 1980 to 2099. If a leap year happen during this valid period, the Date function will correctly return the 29th day as the last day of February. Therefore, missing the divisibility by 400 needs not be fixed in the next few decades, unless some historian or archaeologist needs to roll back the date to 1900 when running a special-purpose application. For most business applications, the leap-year problem can wait until the year-digit problem is fully resolved. The focus of our discussion is therefore on resolving the year-digit problem.

3. The Sources of Year-Digit Problem

The year-digit problem may come from many different sources: system, applications, data, users, and networks (see Figure 1). Our discussion focuses on the platform of IBM PC. Large mainframe systems have very similar problems. The discussion in this paper can be applied to any other computer platform experiencing the same year-digit problem. The proposed solution process in the later section is useful in combating any year-digit problem beyond the year of 2000.



Legend: <------ = this call is not a recommended practice.
<------ = this call is made only once to initialize or synchronize the clock.
<------ = this call is made as many times as necessary.

Figure 1. The Sources of Y2K Problems

3.1. System

The system component includes both hardware and software. Every personal computer contains two clocks: the real-time clock (RTC) and the system

clock. While the system clock runs only when the system is powered on, the RTC keeps ticking and is powered by a low voltage (typically 3- to 5-volt) battery when the system is powered off. The RTC is a built-in hardware clock residing on a CMOS (complementary metal oxide semiconductor) chip and only keeps the time of day within a 100-year calendar. Although the RTC uses a four-digit year, it does not update the century digits. The value of century digits ("19") is stored in one byte on the nonvolatile memory of CMOS and cannot be changed by the RTC. When the RTC rolls over into the year of 2000, the two-digit year value simply overflows into "double zero." With the other two digits of century value remain unchanged, the RTC is now showing a year of 1900, thus is a cause of year-digit problem.

In contrast to RTC, the system clock on the IBM PC is a virtual clock because it is not really a clock but a 24-hour timer. While the RTC may actually use an external quartz crystal oscillator to track the time and date, the system clock uses a programmable interval timer (such as Intel 8253 chip) that merely increments a counter 18.2 times every second [5].

When the computer is turned on, the system clock is synchronized with the RTC. These two clocks run independent of each other when the computer is up and running. After the computer is booted, the host operating system (such as MS-DOS) reads the system clock's counter via the BIOS (Basic Input/Output System) and converts it into hours, minutes, and seconds. Regarding the date, the host OS reads it from the RTC via the BIOS during initialization, then tracks the date independently based on the clock's rolling over at midnight [3]. An older (before 1977) computer, in contrast, must read the time and date directly from the RTC because it does not have a BIOS chip. The BIOS concept was introduced in 1977 and implemented in 1984 on the IBM PC/AT. It is a read-only memory (ROM) chip that resides permanently in PCs and contains instructions for a standard interface between software and hardware. It also uses the CMOS memory as a storage place for the hardware configuration information that it manages.

Finally, a guest operating system such as Microsoft Windows is running on top of the host operating system, i.e., MS-DOS. It uses and expands the functionality of the host operating system. It works the same way as the host operating system does in tracking the time and date. However, it usually has more format styles of displaying the date value and might have a range of valid date different from that of the host operating system. For example, the valid date of MS-DOS ranges from January 1, 1980 to December 31, 2099 and its format styles are mm/dd/yy and mm/dd/yyyy in the U.S. Windows 98 has the same range of valid dates and can display the date in many different format styles. Of all the aforementioned four different clocks (RTC, system clock, host OS clock, guest OS clock), the RTC is the most accurate one, thus is the best alternative for a time-and-date critical application.

3.2. Applications

An application is the software that automates a specific group of related business processes. It is usually running in conjunction with a language processor (e.g., C, COBOL, Visual Basic, Java, HTML, etc.) or a database software (e.g., Microsoft Access, xBASE, Oracle, DB2, etc.). The major functions of an application are to read, write, calculate, compare, transform, and organize the data on the system. Most applications request the current time and date from the operating systems. Some request them from the BIOS and very few directly from the RTC. Therefore, the valid dates for application software are identical to those of the system software. For example, the Visual Basic running on MS-DOS has a range of valid dates from January 1, 1980 to December 31, 2099, while the COBOL85 running on IBM-AIX starts the valid date from January 1, 1601. Assuming these external sources provide accurate time and date in the year of 2000, the year-digit problem lies only in the width format of the date variable. If the "year" format is defined with two digits, year 2000 will be represented by "00" and thus it is a cause of year-digit problem. However, not all applications will have serious consequences, unless the year values are used in logical comparisons or mathematical operations.

3.3. Data

The data sources usually include the external read-write storage devices, such as disks and tapes, and the hard-copy source documents. The source documents include standardized forms and turnaround documents; both contain hand-written data and/or computer-printed data. A turnaround document is typically implemented as a form and eventually reentered the system as an input (e.g., invoice payment slip, deposit slip, etc.). Both hand-written and computer-printed data must be re-entered into the external storage before being processed by the software applications. If the year value on these storage media is kept only in two digits, both year 1900 and year 2000 will have a value of "00" and thus it is a cause of year-digit problem. In fact, most application systems use four or more digits to process

year value on its internal hardware components. Therefore, external input is the major source of year-digit problem, other the system clock when running business applications.

3.4. Users

A user usually supplies the application systems with year value data via a video display terminal (VDT) or a hand-written source document. By today's standard, a VDT is equivalent to the keyboard and monitor of a personal computer system. The year-digit error on the user's part is actually created by the designers of the screen format on the VDT or the printed format on the source-document. If a user is presented with two-digit space for year value on a form or a screen, he or she will supply "double zero" for the year of 2000. This is a cause of year-digit problem and must be fixed before asking the user to supply the year value. Nonetheless, if the data entry screen requires four digits of year value but the source form recorded only two digits, a typical user should be intelligent to add the century value to the 2-digit year and enter four digits onto the screen. Therefore, users are the least likely source of year-digit problem if the data entry applications are designed properly.

3.5. Networks

A personal computer may be a client to a network server. During the network operating system login, the network server may reset the time and date of a client to insure consistent operation among many clients. If the year digits of the server has the "double zero" error, all the clients will inherit the same error. Furthermore, an application running on a network may receive or read a "double zero" year value from the other applications or data sources on the network. Both of these are possible causes of year-digit problem.

Likewise, a data storage media being shared by the applications running on a network may receive a "double zero" year value from the network when the applications running on the network saves or prints the year value. Then, the content of a data storage or document will contain invalid year value and this is a cause of year-digit problem.

4. A System Approach to Classifying Year-Digit Problem

Given the aforementioned sources of year-digit problem, one may use the system approach to classify and prioritize the potential problems faced by an application system. According to the system approach, an application system should consist of four stages: input, process, output, and feedback/control. The year-digit problem may occur in any one of these stages as its value flows through the system. The possible year-digit problem at each stage is briefly described below.

4.1. Input Stage

There are two types of input: *internal input* and *external input*. The internal input of year digits includes the *hardware RTC* and the *date-assignment statement* or the *date-initialization statement* in the program code. The possible types of external input to the system include *data entry*, *file retrieval*, *database retrieval*, and *system interface*. A data entry is originated from a source document and year digits are prescribed by the number of blanks left on a document that can be entered by a user. File or database retrieval, on the other hand, is originated from a program call and prescribed by the number of year digits stored on the input file or database. Finally, a system interface is originated from passing a year value from one program to another, either locally or through the network, and prescribed by the formats of the values at both ends of the interface.

The year digits of a data entry can be easily identified. Likewise, the year digits on a database can be identified with reasonable effort by examining the database schema or dictionary to identify the database items that contain year value. On the other hand, identifying the files that contain year digits is not so easy. One must review all the format statements in the program code that read data from files. As for the other types of problem, the RTC problem is well-known and easy to fix, the internal date-assignment or date-initialization statement and the system interface problems are very difficulty to pinpoint.

4.2. Process Stage

Data in the system process stage may be operated on mathematically or logically. The mathematical operation on year digits may involve increasing or decreasing the values of year digits (e.g., YEAR2 = YEAR1 + 10; YEAR0 = YEAR1 - 5) and calculating elapse time between two years (e.g., AGE = YEAR2 - YEAR1). As for the logical operation on year digits, it may only involve the "IF" statement that uses year or date value as the condition. For example, IF (CURRENT_DATE < "01/01/00") THEN {…approve the credit…}; IF (CURRENT_DATE = "01/01/2000") THEN {…release the prisoner…}; where CURRENT_DATE comes from the system clock. The first statement will not work if the CURRENT_DATE is 11/30/1999 unless the "00" is

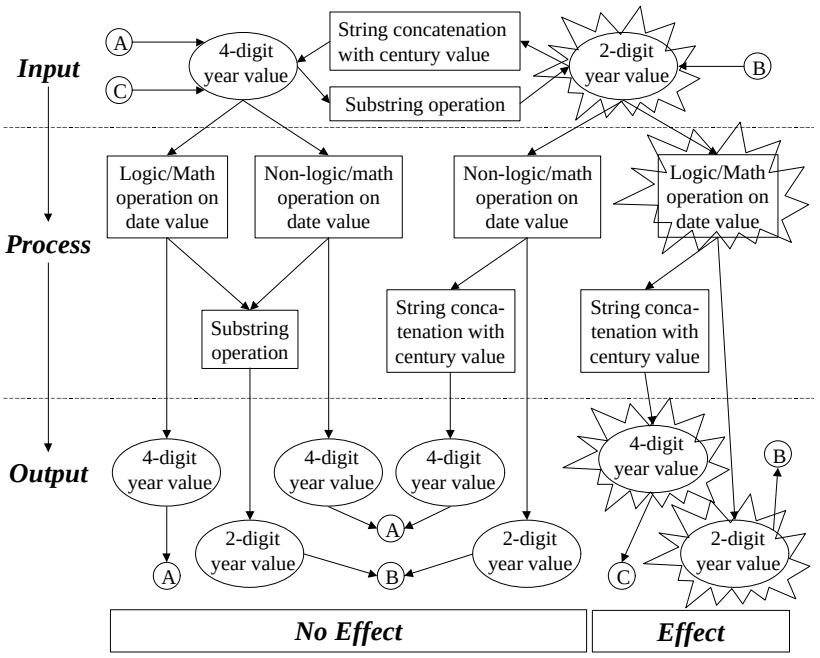
interpreted as "2000." The second statement will work since the system clock will correctly roll over to the year of 2000. Although there are many applications running on a computer system, the year-digit problem has effect only on certain type of applications. Figure 2 shows only those use two-digit year values as input and process them with logical or mathematical operations are problematic applications. All others are free from errors. These applications that have the effect are what we need to worry about. Figure 3 provides the examples of the effected and non-effected application processes.

4.3. Output Stage

The types of output include internal output and external output. The internal output is the work-in-process value of a job execution temporarily stored on an internal memory device. The number of year digits on the internal output is determined at the input and process stages, therefore, resolving the problem at the input and process stages automatically resolve the internal output problem at this stage. In contrast, the external output from the system is more complicated. It includes printed output and electronic output. A printed output may be a form or a document. An electronic output can be a screen shot, a form, a document, a file, or a database. The year-digit problem on an external output mostly is easy to identify, except those on an electronic data file or database that must be identified by examining program code or database schema.

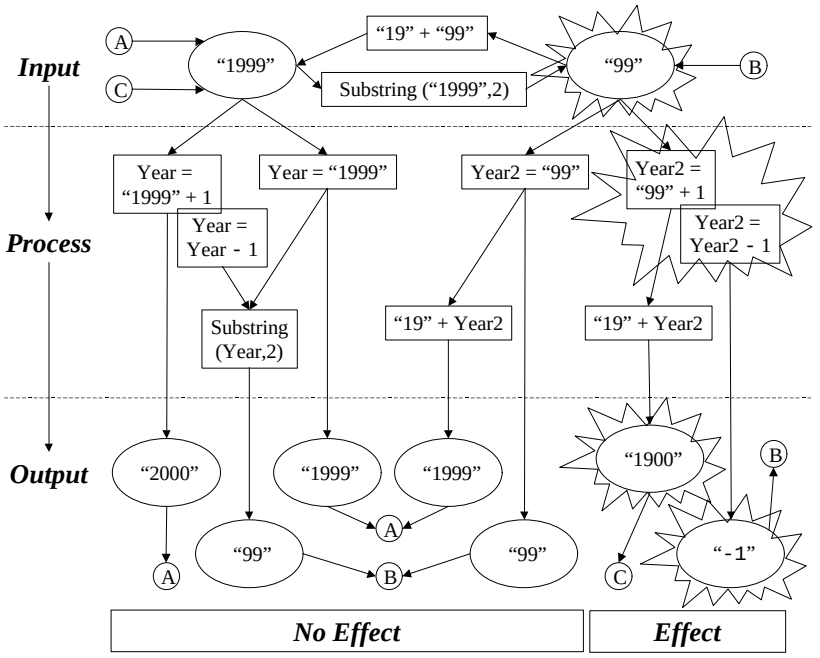
4.4. Feedback Stage

The feedback is actually a special type of process. It is used to determine the course of action when an exception occurs. There are two types of feedback and action: *manual feedback and action* and *automatic feedback and action.* Manual feedback and action are performed by a human being and he or she should be intelligent enough to automatically adjust the year digits and resolve the year-digit problem. Automatic feedback and action are typically implemented in the process stage and



Legend: 1. The symbol of explosion indicates the source of year-digit problem.
2. The two rectangles placed in the input stage belong to the process stage.

Figure 2. Classifying the Applications into Effect and No-Effect Groups



Legend: 1. The symbol of explosion indicates the source of year-digit problem.
2. The two rectangles placed in the input stage belong to the process stage.
3. Year2 is a variable with fixed format of two digits.
4. Substring(Year,2) is a function that returns two right-most digits of the Year value.

Figure 3. Examples of Effected and Non-Effected Applications

Table 1. Types of Y2K Errors and Their Remedies

	Error Description	Remedy
Input	*Easy-to-Fix Errors:*	
	I-1: Century value cannot advance automatically on the CMOS real-time clock.	I-1: Replace the BIOS; or install a software patch; or manually run the BIOS setup program to advance the century value.
	I-2: Year digits on the CMOS real-time clock overflow to"00."	I-2: Same as I-1 above.
	I-3: Fewer year digits on the source documents and data entries.	I-3: Review all source documents and redesign them to require entering 4 digits of year value. If data entries are voluminous, modify the program to automatically add the default century value when users enter 2 digits of year value.
	I-4: Fewer year digits on the input databases.	I-4: Review all database structures and redesign them to store 4 digits of year value.
	Expensive-to-Fix Errors:	
	I-5: Date initialized with 2-digit year value in the source code.	I-5:* Review the in-house source code in all applications to initialize 4 digits of year value.
	I-6: Date assigned with 2-digit year values in the source code.	I-6:* Review the in-house source code in all applications to assign 4 digits of year value.
	I-7: Fewer year digits on the input data files.	I-7:* Convert all physical files to store 4 digits of year value, then identify all applications that use these files and change their file-format code with 4 digits of year value.
	I-8: Some networks initialize the clients with 2-digit year values.	I-8: Update all erroneous network clock components and operating systems; or install a software patch on all network servers and clients to concatenate proper century value with the year value.
	I-9: Some programs pass 2-digit year values to other programs.	I-9:* If the program passes year value by variable, resolving I-5 to I-7 will resolve this error; if it passes by constant, change all such programs in all applications to have 4 digits of year constant.
Process	*Easy-to-Fix Errors:*	
	P-1: Erroneous leap-year calculation performed by CMOS real-time clock.	P-1: This error rarely occurs. If it does occur, replace the old CMOS with an error-free new CMOS.
	P-2: Erroneous leap-year calculation performed by system software.	P-2: This error rarely occurs. The error usually occurs only when a century date is not divisible by 400. If this occurs, request the software vendor to replace it with an error-free version.
	P-3: Erroneous leap-year calculation performed by application software.	P-3: Same as P-2 above.
	Expensive-to-Fix Errors:	
	P-4: Erroneous leap year calculation in the in-house source code.	P-4:* This error rarely occurs. The error usually occurs only when a century date is not divisible by 400. If this occurs, review and correct the source code of all applications that keeps track of leap years. The source code may be an independent module itself or a part of a module.
	P-5: Incorrect sorting due to 2-digit year values.	P-5:* Resolving input errors of I-1 to I-8 will resolve this error.
	P-6: Predicate statements use 2-digit year value as the condition in the in-house source code.	P-6:* Resolving input errors of I-1 to I-8 will resolve half of this error. Then identify the predicate statements in all applications that use 2-digit year constant and change the constant to a variable containing 4-digit year value.
	P-7: Mathematical formulas operate on 2-digit year value in the in-house source code.	P-7:* Same as P-5 above for mathematical formulas.
Output	*Easy-to-Fix Error:*	
	O-1: Fewer year digits on the printed or electronic (normal or turnaround) documents, forms, and/or screens.	O-1: As long as every year-value data entry requires 4 digits, the only concern is on the electronic turnaround documents and screens that will be read directly by the computers. Review all electronic turnaround documents and screens; and change year value to 4 digits.
	Expensive-to-Fix Error:	
	O-2: Fewer year digits are stored on the electronic output databases.	O-2:* In addition to I-4 remedy, review the source code in all related database applications to make sure the output contains 4 digits of year value.
	O-3: Fewer year digits are stored on the electronic output data files.	O-3:* Same as I-7 above.
Feedback	*Easy-to-Fix Errors:*	
	F-1: Manual feedback using 2-digit year value as a control variable for course of action.	F-1: This error can be easily resolved by the feedback recipients. Besides, resolving input and process errors above will resolve this error.
	F-2: Manual date stamp with 2-digit year value on the feedback signal or data.	F-2: No need for changing hand-written date stamp. For electronic date entry, redesign the date entry field in the source code to require 4 digits of year value as screen input and as file output.
	Expensive-to-Fix Errors:	
	F-3: Automatic feedback using 2-digit year value as a control variable for course of action.	F-3:* Resolving input and process errors above will resolve this error.
	F-4: Automatic date stamp with 2-digit year value on the feedback signal or data.	F-4:* Review the in-house source code in all related applications and redesign the date stamp field to require 4 digits of year value as system input and as printed or file output.

- indicates this remedy requires extensive review of in-house source code in all applications.

performed by the system itself through comparing the actual output with the expected output. Input and process are then automatically adjusted by the system to produce the expected output. This type of feedback is common in embedded systems such as artificial neural networks, missile control systems, and energy control systems, etc. Another type of automatic feedback is by comparing the existing condition with an pre-established condition to determine the course of action. This is common in business systems such as automatic inventory replenishing system, automatic fund transferring system, etc.

Two-digit year value may cause serious problems in an automatic feedback system when it is used as a control variable in determining the course of action. For example, let's assume that we are charging a purchase with a credit card having an expiration date of 01-23-01. If the credit-card system is not Y2K compliant, it may feedback the message saying "Credit card has been expired." Fortunately, this error causes no harm but inconvenience. Without a valid expiration date, our transaction will not be approved and we cannot make the purchase with the credit card, although the salesperson realizes we are holding a valid one.

Another problem occurs when date stamp is used for control purpose. If 2-digit year value is used in the stamp, it is most likely outputted to a data storage. When this year value is retrieved by another application, it becomes the faulty data in the input stage as shown in Figure 2. To resolve this problem, a 4-digit year value must be used in an electronic date stamp.

5. Solving the Y2K Problem

5.1. Types of Errors and Their Remedies

To solve the year-digit problem, one may start with the easy tasks and proceed to the difficult ones. In our opinion, one should start with the input stage, followed by the output stage, the feedback stage, and finally the process stage. Table 1 enumerates the errors identified in the last section. These errors are further classified into two categories: easy-to-fix and difficult-to-fix errors. The possible remedies to these errors are identified as well in this table.

5.2. Tackling the Errors in Applications

There are several remedies listed in Table 1 that require extensive review of in-house source code in all related applications. This is why the Y2K solution effort becomes very expensive to complete. It is then necessary to formulate strategy for tackling these errors. The first thing we must do is to resolve the errors that are coming from the runtime environment outside the application itself. These include installing Y2K-compliant CMOS, RTC, operating systems, and vendor's application software on the network servers and clients.

After we isolate the errors from the in-house applications, we must decide on where to start fixing our applications. Table 2 indicates the two dimensions one should consider: *effect* and *criticality*. The best progression is to start with the mission-critical applications that are affected by the year-digit problem (Category I). After the problem has been corrected for these applications, we may proceed with those affected but non-critical applications (Category II). As for no-effect applications (Categories III and IV), there is no immediate need to change the year digits. We may resolve these problems after we have completed resolving the errors in Categories I and II.

Table 2. Classification of Applications

Application	Effected application	Non-effected application
Mission-critical application	I	III
Non-critical application	II	IV

To begin tackling effected applications, the best way is to follow the trail of input, process, and output of the effected applications as indicated on the right side of Figure 2. If we have a system that is well documented, it would be easy to narrow down specific input and documents, data files, or databases, and special program module that need to be fixed. If our system documentation is poor or does not exist at all, we could follow the ***divide-and-conquer approach*** to resolve the problems. Under this approach, each user department should establish a committee to review all its printed and electronic source documents, data entries, data tables, and output documents. The focus is to find the data on the forms or documents that might be susceptible to year-digit error. This includes the field whose value might be calculated based on date value, e.g., interest, due date, late charge, etc. The departmental committee then submits a report to the information systems department (ISD). In this report, visible errors and potential errors of certain input and output forms or documents are identified, and the impact of these errors are described and prioritized. The ISD takes this report to identify critical applications and examine the in-house source code of these applications. Since any error found at the application level needs to be traced down to the module level, we

strongly recommend reviewing, fixing, and testing the source code at the module level.

According to Figure 2, any program module that contains no logical or mathematical operation on year value need not be fixed. Before putting away such non-effected program module, we should review the source code to see if the module calculate for leap years. Watch for the constant value of "4" or "400" in the source code. Record and report if either the divisibility by 4 or the divisibility by 400 is missing in the source code before we put away the non-effected module. If our application needs to roll back the year to 1900 or roll forward to 2100, we must immediately fix the problem and ensure that both leap-year requirements are implemented in the source code. Otherwise, the leap-year problem can be fixed later.

If the module contains logical or mathematical operations on year values, make sure the operations does not involve 2-digit year constants (e.g., 06/01/99). If anyone does, change the constant to a variable containing 4-digit year value using a date initialization statement. In fact, the variable is flexible enough to accept any number of digits for year value as long as it is within the allocated memory limit. Once all operations on year values contain no year constants but year variables, we may focus on the logical operations in an application. Using the ***gatekeeper approach***, any year variable entering the logical operation must contain 4-digit year value. If not, we can trace back where the value comes from and fix the error. During the tracing, we might run across date calculation errors. We should fix these errors at the same time. This approach allows us to switch our focus to identifying faulty 2-digit or 4-digit input originated from the faulty output of the effected module as seen in Figure 2.

When dealing with input and output, two axioms come to mind: "Garbage in, garbage out." and "Where there is a file output, there is a file input." Even on the non-effected side in Figure 2, a faulty input ("garbage in") will definitely produce faulty output ("garbage out"). This faulty input is most likely coming from a data storage, not from a user. That is, some applications out there are creating faulty outputs on data storage, which will be read into other applications electronically. Yet, these faulty outputs are originated from some faulty inputs, as shown on the right side of Figure 2. Consequently, solving the year-digit problem in the input stage should resolve most of the existing Y2K problems. As discussed earlier, these problems include:

- hardware RTC,
- date-assignment code statement,
- date-initialization code statement,
- data entry,
- file retrieval,
- database retrieval,
- system interface.

For the remedies of these problems, please refer to Table 1. Remember earlier we had our user departments to submit impact reports. These reports may be used to identify data entry screens, source documents, file structures, and database structures that need to be re-designed. As for the leap-year problems in effected applications, the same way of handling those in non-effected applications may be applied. One final note is that the solution process described above requires the source code be available. If this is not the case, i.e., we only have the executable code, the first thing we must do is to run the code with Y2K test cases on a Y2K-compliant machine and network. If we are not able to do so or any Y2K error reveals, we might need to re-write the entire code from scratch.

6. Conclusion and Recommendations

The Y2K problem consists of the year-digit calculation problem and the leap-year calculation problem. These problems are straightforward and easy to identify. However, the problems may exist in countless lines of code written since the advent of computer. The best way solve these problems is to identify effected and mission-critical applications. An effected application is one that has a 2-digit year value as the input and has logical or mathematical operations processing this year value. A mission-critical application is one that has great and irrevocable impact on a business. We should start with fixing the problems exist in these applications. To do so, we should embrace the divide-and-conquer approach and focus on the module level. Since any error found at the application (top) level needs to be traced down to the module level, it is more effective to review, debug, and test the source code at the module level. Coupling with the gatekeeper approach, we can further narrow down our focus to faulty input and reduce our Y2K effort.

We have learned a painful lesson from such a mindless mistake. If we could start all over again, we could have taken the following actions. In fact, these actions can mitigate the year-digit problem beyond the Y2K compliance when the year value requires more than four digits in the year 10000 (Y10K).

1. **Migrate to database environment.** Under the database environment, the date field on a database file can be share by many different applications. Unlike the legacy file system, the code of each database application needs not specify the date format or the number of year digits. If the number of year digits increases, all we need is to change the database structure. There is no need to change any code in any database application.
2. **Standardize names and formats of our own date-related variables.** The names and formats of the date variables provided by our system or application software are somewhat different. We must know their differences. If we code our own date-related variables, we should use the complete word "date" as part of the variable name. This allows us to search the text ("date") and easily determine whether the code processes any date-related variables. As for the formats, they must be standardized across different files. A 4-digit format for century year value is recommended.
3. **Institute standards in programming date-related variables.** Do not use constant value of date to compare with conditions or calculate in a formula. Use an initialization or assignment statement to set the value of the date-related variables. This allows us to search the text of "date" and identify the constant value. If possible, we should change the code to read the value from an external file (e.g., an ".INI" file) or a database file. Although external file or database access will degrade the system performance, user should feel no difference in performance sine initializing or assigning values to date-related variables only happens once during the runtime of a code module or even the entire application.
4. **Maintain good documentation on the usage of date variables.** During our Y2K compliance effort, if we knew which modules process date-related variables, we could have easily identified and fixed the effected code modules. Better yet, we could keep a list of effected modules; only these modules need to be fixed when the number of year digits grows.
5. **Know our mission-critical applications**. Talk to the users to identify the impact of each application during the off-peak hours. In fact, the impact report should be a part of the feasibility-assessment report when a new application is being proposed. Prioritize the impacts and keep a list of mission-critical applications. When the year-digit problem happens again, we will know exactly what applications need to be looked at and fixed right away.
6. **Do not throw away our source code**. Otherwise, if the year-digit problem resurfaces, we will not be able to fix it. We might end up re-writing the code starting from scratch. The problem is that when we threw away the source code, we most likely threw away the documentation of the application as well. No one can tell exactly what makes this application tick without the documentation. Chances are when we rewrite the application, we might miss something that causes other applications to fail. Eventually, we might end up with much more than what we bargained for.
7. **If we write our own date and leap-year calculation routines, create library modules for all other modules to access.** These library modules are to provide date and leap-year calculation functions as well as current time, day, year, and century information. In object-oriented terms, a date object can be created to provide all these functions and information items. If there is any error in date or leap-year calculation, this practice releases us from reviewing the source code of all applications. All we have to do is to fix the code in these library modules or the date object.
8. **Educate users and programmers.** The Y2K compliance effort is a joined effort between users and programmers. As new users and programmers get on board, they must know the impact of the year-digit and leap-year problems. They must be educated to follow the recommended practices described above.

References

[1] Duffy, Daintry "Field Report: World War II Still Wins," *Software Magazine*, Vol. 18, No. 6, April 15, 1998, p. 6.

[2] Hayes, Ian "Dateline 2000: Think Globally, Volunteer Locally," *Software Magazine*, Vol. 18, No. 13, October 1998, p. 76.

[3] National Software Testing Laborary, “Background on the Year 2000 Issue,” NTSL, Inc. 1998, <http://www.ntsl.com/ymark2000_background.html>.

[4] Noone, Steve "ITS Cleans Computer Code for Year 2000," *Mustang Daily*, California Polytechnic State University, San Luis Obispo, Tuesday, March 9, 1999, pp. 1 & 7.

[5] Sargent, Murray, III and Shoemaker, Richard L. *The IBM Personal Computer From the Inside Out*. Reading, MA: Addison-Wesley, 1984.

[6] Wong, Edward "CalPERS Year 2000 Project Achieves Major Milestone," *PERSPECTIVE: A California Public Employees' Retirement System Newsletter*, Winter 1998-99, pp. 1, 3.

Eldon Y. Li is Professor and a former Coordinator of MIS area at the College of Business, California Polytechnic State University, San Luis Obispo, California, U.S.A. He holds a bachelor degree from National Chengchi University in Taiwan and M.S. and Ph.D. degrees from Texas Tech University. He has provided consulting services to many firms for a variety of software projects and served as a management consultant to the clientele of the U.S. Small Business Administration. He was a software quality specialist at Bechtel Corporation's Information Services Division and a visiting software scientist at IBM Corporation. He is a Certified Data Educator (CDE) and is Certified in Production and Inventory Management (CPIM). His current research interest lies in human factors in information technology (IT), strategic IT planning, software engineering, quality assurance, and information and systems management.

Disclaimer: The purpose of this article is to provide a self-guided tour through the journey of Y2K-compliant effort. Both the author and the publisher make no guarantees or warranties as to the accuracy, completeness, or suitability of information provided in this article, and assume no responsibility whatsoever for any consequences whatsoever resulting directly or indirectly from your use of this information. Users of this article are responsible, on their own, to determine the accuracy and suitability of all information contained in this article.