The Year 2000 Problem: Issues and Implications

Dennis B. Smith
Hausi A. Müller
Scott R. Tilley
April 1997

TECHNICAL REPORT
CMU/SEI-97-TR-002
ESC-TR-97-002

Technical Report
CMU/SEI-97-TR-002
ESC-TR-97-002
April 1997

The Year 2000 Problem: Issues and Implications

Dennis B. Smith
Software Engineering Institute

Hausi A. Müller
University of Victoria

Scott R. Tilley
Software Engineering Institute

Reengineering Center
Product Line Systems

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

This report was prepared for the

SEI Joint Program Office
HQ ESC/AXS
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

(signature on file)

Thomas R. Miller, Lt Col, USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number F19628-95-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 52.227-7013.

This document is available through Research Access, Inc., 800 Vinial Street, Pittsburgh, PA 15212.
Phone: 1-800-685-6510. FAX: (412) 321-2994. RAI also maintains a World Wide Web home page. The URL is http://www.rai.com

Copies of this document are available through the National Technical Information Service (NTIS). For information on ordering, please contact NTIS directly: National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161. Phone: (703) 487-4600.

This document is also available through the Defense Technical Information Center (DTIC). DTIC provides access to and transfer of scientific and technical information for DoD personnel, DoD contractors and potential contractors, and other U.S. Government agency personnel and their contractors. To obtain a copy, please contact DTIC directly: Defense Technical Information Center / 8725 John J. Kingman Road / Suite 0944 / Ft. Belvoir, VA 22060-6218. Phone: (703) 767-8222 or 1-800 225-3842.]

Table of Contents

Acknowledgments

We wish to thank Shawn Bohner of MITRE-Tech, Mike Olsem of the Software Technology Support Center (STSC), and Tom Driscoll of Formal Systems. They provided us with a number of insights and generously shared their time by permitting us to interview them.

We also wish to thank Archie Andrews, John Bergey, Dave Carney, Paul Clements, Linda Northrop, Doug Waugh and Nelson Weiderman of the Software Engineering Institute, Kenny Wong of the University of Victoria, and John Salasin of DARPA who provided excellent comments on earlier versions of this paper.

This work was supported in part by the U.S. Department of Defense.

The Year 2000 Problem: Issues and Implications

Abstract: A lot of attention has recently focused on the possibility that a great deal of software will fail at the turn of the century because of the way dates are stored and processed by computer programs. Attitudes range from alarmist to unconcerned regarding the magnitude and implications of the problem. This report outlines the basic issues of the so-called "Year 2000" (Y2K) problem and discusses some of its implications.

1. Introduction

This report is addressed to chief information officers (CIO), program managers, and project leaders responsible for developing strategies to address the Y2K problem. It outlines the basic issues and core technical tasks required for Y2K efforts, highlights the types of technology available, and identifies some of the gaps in technology. Our findings are based on a review of published literature, documentation available on the World Wide Web, and interviews we conducted with individuals who have already been active in addressing Y2K issues. The focus of this report is on issues and implications of the Y2K problem that are broadly applicable to most organizations.

The next section provides a description of the Y2K problem. Section 3 presents important planning issues that need to be addressed before a technical solution to the problem is chosen. Section 4 discusses the core technical tasks that require attention. Section 5 identifies the gaps in current technology related to the Y2K problem. Section 6 outlines the requirements for a coordinated effort needed to solve the Y2K problem. The appendices provide pointers to resources and a list of the individuals interviewed.

2. Problem Description

The crux of the Y2K problem is simple. The overwhelming base of installed legacy systems have routinely stored and processed dates without a century indicator. Date fields are typically stored in "MMDDYY" format, where "YY" represents the last two digits of a given year. This type of two-digit date encoding will probably fail to recognize "00" as representing the year 2000 and instead will erroneously interpret "00" as representing 1900, since the "19" has been implicitly assumed. There is an additional problem that, contrary to popular belief, 2000 is a leap year based on the rule that it is divisible by 400 and this rule is not commonly programmed into existing systems.

The date problem is crucial because dates are fundamental to the calculation of algorithms for virtually every type of electronic support system, such as command and control, assembly lines, interest and loan payments, invoices, airline reservations, and so on. Since computer systems affect almost every aspect of modern life, it is apparent that the electronic world that we take for granted could be severely crippled if the dates are not corrected.

Although the bulk of current attention is focused on the turn of the century, the problem can actually manifest itself at various times. For example, credit card renewals and motor vehicle license renewals are usually provided for several years, while systems dealing with bond issues and life insurance policies have already faced the problem. In each application, it is necessary to determine when the problem will first appear and develop plans to deal with it before the first critical date.

The Y2K problem is urgent because it has a hard, non-negotiable deadline. Many applications are at risk if the problems are not corrected before the application-specific critical events. The problem is severe because of the following five reasons:

1. **Pervasiveness**. Dates are present in some form in every type of application, including information systems, command and control systems, and weapons systems. They form the basis of derived calculations for such data as age and amortization schedules. Dates are also the basis for a number of sorting routines, error checks, validations, and logic for terminating applications.
2. **Idiosyncratic coding practices**. Date fields have been coded and used idiosyncratically by generations of programmers. Dates are sometimes hard-coded in programs, passed as parameters, aliased, redefined through program algorithms, or hidden in job control language. The identification of all date fields within a single application is a difficult task. There is not a single approach that will automatically identify all instances of dates without a large number of false positives or false negatives.
3. **Poor maintenance and coding practices**. It is well known that documentation often becomes obsolete in legacy systems. This problem is compounded for older legacy systems that were developed with far less discipline than current systems. Often the original programmers and maintainers have moved on long ago. Even more discouraging are our interview results suggesting that, for some organizations, up to 20% of their source code may be inaccessible. In these cases, the source code will need to be rebuilt, reengineered, or replaced.
4. **Complexity of computer systems**. Most existing applications depend on a number of

support systems such as databases, networks, and telecommunication monitors. They rely on commercial compilers and run on commercial operating systems, with specific hardware platforms. The date problem needs to be handled correctly and consistently not only within the application, but also within all the support systems and interfaces to which it is connected.

5. **Size of the legacy base**. The sheer size of the installed base of legacy programs and support software adds an additional dimension to the date problem. Worldwide, there are literally billions of lines of legacy code, written in over two thousand languages, accessing hundreds of databases, and running on an ever evolving variety of operating systems and hardware platforms. Because it is necessary to make these changes for almost every current system, the necessary corrections will require a concerted, if not heroic, effort.

These five factors interact to turn a conceptually straightforward task into an almost insurmountable challenge. To address the problem successfully requires strong automated support as well as substantial human intervention.

3. Planning Considerations

Before choosing a technical solution to the Y2K problem, important planning issues need to be addressed. The first is awareness: raising the Y2K problem to a sufficiently high management level so that it gets attention. The second is estimating the cost of solving the Y2K at your organization. The third is deciding which systems are critical to your business, and hence deserve top priority, and which systems can be left unaltered for the time being.

3.1 Awareness and Action Planning

Raising awareness of the Y2K problem within your organization is an early critical success factor. Although knowledge of the potential existence of the Y2K problem has been common within software engineering circles since at least the 1970s, a number of organizations have been slow in addressing the problem [CGRO 96, Cassell 97, Kappelman 96]. There are various reasons why this is the case, such as

- the normal aversion to long-range planning
- the belief that the problem will occur on somebody else's watch
- a naive sense that it will not be severe within a particular organization
- an inability to scope the local impact and develop a plan to resolve the problem

It seems that high-level managers tend to underestimate the magnitude of the problem within their own organizations, and they initially assume that a relatively low-intensity effort will resolve it. In fact, the resolution of the problem requires an intensive and time-consuming effort that is fully funded and supported.

A respondent from our interviews reported that he has offered to conduct a free pilot analysis of a system, to determine the extent of the problem on a sample of a million lines of an organization's code. In every case, the actual number of problematic lines of code has been several orders of magnitude higher than the prior estimates made by the CIO of the organization.

These experiences suggest that key critical success factors for addressing the problem include

- early specific awareness of the impact of the problem within the organization
- early management awareness and involvement
- careful planning, estimation, and allocation of resources

3.2 Cost Estimation

Several broad, order-of-magnitude estimates have been made on the cost to fix the Y2K problem. The Gartner Group [Cassell 97] estimates the worldwide cost to range from $300-

$600 billion, at a rate of $1.10 per executable line of code (LOC). A study by MITRE [Roberts 96] made higher per LOC estimates for the resolution of the problem within the US Department of Defense, which has a greater percentage of embedded systems. This study made estimates by type of system as follows:

1. ground and airborne radar systems: $2.02 to $8.52 per LOC
2. communications processing systems: $1.23 to $5.54 per LOC
3. C2 planning systems: $1.02 to $1.84 per LOC
4. logistics support systems: $1.02 to $1.39 per LOC

These estimates will obviously need to be refined as more organizations develop experience in addressing the Y2K problem.

3.3 Triage Planning

For a number of organizations, it will not be possible to resolve the Y2K problem for all their systems before the systems fail, either on 01/01/2000, or before that date, depending on the application. In these cases, hard decisions will need to be made. Martin [Martin 97] and de Jager [de Jager 96] recommend a triage approach that sets priorities for selecting which systems to convert and save, and which to allow to fail.

As part of the initial plan, business goals and objectives need to be defined. Some of the issues to be addressed from a business perspective include an estimation of the business value of each system, an analysis of the risks and liabilities if the system would fail, and plans and requirements for future enhancements, new business needs, and potential replacement of existing systems.

In the triage approach, mission-critical systems that are not slated for immediate replacement need to be converted regardless of the cost. Systems of low strategic importance may be allowed to fail if resources are unavailable. Decisions can be made on which medium-importance systems to convert. These decisions can be based on business value, cost, difficulty of conversion, technical risks, resources available, and time remaining. These factors need to be revisited frequently to track the progress of Y2K projects and to reallocate resources to account for current progress and risks. This approach enables rational decision making as constraints increase.

For systems that are not to be converted, contingency plans need to be developed. These plans develop an assessment of the functionality of the system, its business uses, interfaces, and financial, technical and legal impact. Options need to be developed for working around the system. Plans also need to be developed for the eventual conversion or redevelopment of the system, as resources become available in the future.

The systems that are to be converted require remediation planning. The remediation plans allocate resources to the task, and develop a schedule for the major technical tasks required. The schedule needs to be monitored closely because slippage can have serious impact on other Y2K conversion projects.

4. Core Technical Tasks

A number of groups have developed a baseline identification of the phases and technical tasks required to resolve the Y2K problem [Bohner 96, Martin 97, IBM 96, STSC] There is an overall commonality in the treatment of the Y2K problem by any organization. However, a Y2K conversion effort can be analyzed from a number of different perspectives—for example

- focusing on the definition of the process for addressing the Y2K problem [Bohner 96]
- evaluating tools and serving as technical advisors [STSC])
- developing data for costs, analyses of the major issues, and a tools catalog [IBM 96, STSC, Martin 97]
- developing a database for Y2K compliance [DISA 97]

We have chosen to focus on a set of six core technical tasks, including a high-level description of the technical issues, the type of currently available technology, and gaps in technology. These tasks are the following:

1. Develop a high-level system inventory.
2. Develop an impact analysis.
3. Plan the remediation.
4. Perform the remediation.
5. Test the changes.
6. Migrate to production.

4.1 Develop a High-Level System Inventory

Inventory or portfolio analysis is a critical and often surprisingly difficult step for a Y2K migration project. Inventory analysis includes identification of the following:

- applications and data sets of the organization
- the object code and libraries used to build these applications
- the source code used to build the object code and libraries
- databases and data files used to generate the data sets
- scripts and command files for building applications and data sets
- parameter, set up and initialization files (e.g., for sorting utilities)
- corporate naming convention policies and plan templates
- other auxiliary and ancillary files (e.g., screen maps)

To complicate the task, portfolios usually include multiple operational versions of both code and data. Having a state of the art configuration management (CM) system eases inventory analysis considerably. Identifying source files through a CM system is usually straightforward. However, it can be surprisingly difficult to identify the exact source code that

belongs to a particular application. The object files and libraries can also be readily identified, but locating the associated source files can be quite difficult.

Identifying the source for a library that was not produced in-house can also present problems. The manufacturers of the libraries regularly produce new releases, but in a number of cases, organizations have selectively installed releases. Tracking down older versions is difficult. In addition, migrating to the latest version of a software product can require changes in the operating environment in addition to the specific Y2K changes. As noted above, source code may be missing for up to 20% of applications.

The vendors for commercial-off-the-shelf (COTS) products might not provide source code and only guarantee Y2K compliance for the latest version. Data sets are usually not included in the configuration management. In addition, any code from an outside vendor that has either gone out of business or no longer supports the software presents a special problem that needs to be addressed.

A further complicating factor is that not all applications are under CM control. As distributed systems, client/server systems, decision support systems, embedded systems, and standalone applications have developed over the past decade, control by the central organization has decreased. Moreover, the networks, support software, database systems, and operating systems all have their own set of different releases, creating a potentially complex set of interactions. Since the Y2K problem is pervasive, all applications need to be considered.

The high-level inventory analysis can take from three to six months. The next core task, impact analysis, develops a detailed fine-grained analysis. The planning estimates will need to be further refined after the detailed impact analysis is completed.

4.2 Develop an Impact Analysis

The impact analysis task determines in detail where dates exist in a system, which modules and statements are affected, and major critical paths. Some applications may contain between 40 and 50 different date formats. Often as much as 10% of the source code contains Y2K defects and must therefore be changed. Therefore, to locate and correct defects using data- and control-flow analysis in million-line systems, automation is the key to eventually successful remediation.

In preparation for impact analysis, the source code needs to be parsed and represented, preferably, in a language-independent form. A well-known reverse engineering tool, Reasoning System's Software Refinery has developed a technique for representing code as an abstract syntax tree. This technique enables code from a number of different languages to be analyzed in a common way. The use of a common abstract syntax tree also facilitates transformations, and enables impact analysis tools to locate Y2K defects.

Date fields that occur in different formats can be identified with standard pattern matching tools such as the UNIX grep facility (albeit with limited success). Pattern matching provides analysis of data dictionaries and declaration statements for instances of dates, as well as searches within the code for selected keywords, such as year, month, age, and policy-period.

Pattern matching tools provide general rules for detecting date fields, date literals, and source code that manipulates or calculates dates. However, these rules need to be adjusted for each organization, since each evolves its own naming conventions. The pattern matching activity is by its nature an iterative process. Each organization needs to develop a library of naming conventions and exclusions, storage conventions, types of problems encountered, false positives and false negatives. In addition, typical parameters, such as age, time, and expiration dates, should be included in the library.

Once patterns are found, a more detailed analysis is performed to focus on the ripple effects of the dates, as well as to eliminate false positives. Slicing traces the uses of a data field through the source code, Job Control Language (JCL) and databases. Forward slicing determines the program text that is affected by a particular data field. For example, a forward slice on the field *date* will find that fields such as age, policy expiration and current interest are affected by it. Backward slicing focuses on the program text affecting a data field. Slicing can find the associated statements that depend on a date field rather than just identify other fields. For example, a backward slice on the field *age* will determine that both date of birth and current date affect its value. Slicing requires tool support as well as a human intervention to make decisions on the application-level semantics.

The output of this task is an identification of areas of the system that use dates, as well as modules that are affected by dates. Strong visualization techniques to present the data can aid in understanding of the Y2K impact as well as the correction of the defects.

4.3 Plan the Remediation

Once the magnitude of the problem is analyzed in detail, the next step is to develop a strategy for remediation. The basic options are

1. expansion of the database to accommodate a century indicator
2. development of date “windows” that are processed differently depending on whether the date is early or late in a century
3. compression of the date fields in the database

Although the most complete solution is to expand the database, it is also the most expensive and time-consuming. A number of factors may suggest alternative solutions including time, resources, future plans, estimated life span of the system, and relationship to other systems. Therefore, the remediation approach needs to be carefully analyzed in terms of both technical and business options.

The basic options are highlighted briefly below.

4.3.1 Database Expansion

The expansion of the database involves changing the database records from a 2-digit year "YY" to a 4-digit year "YYYY." As a result, the date February 1, 2000 would be stored (in "YYYYMMDD" format) as "20000201," instead of as "000201" (in the currently typical "YYMMDD" format). A secondary option would be to keep the current date fields, and to add a century indicator as a separate field in the database.

The expansion of the database requires not only the conversion of the database itself, but changes to all programs that reference dates in order to correctly use the new date format. It will also require the development of a number of conversion programs to create the new database format and convert existing records.

The database expansion option is complete and permanent. However, since it requires database changes, changes to all existing date related programs, and the development of a set of conversion programs, this option is the most costly. In addition, there may be a negative impact on performance, as well as the need for expanded disk space. This is usually not an option for archival data.

4.3.2 Windowing

The date window approach keeps the database unchanged with a 2-digit year indicator. Dates are processed through logic that interprets the 2-digit years.

The windowing approach defines a 100-year window and specifies how dates in specific intervals within that window are to be processed. In the simplest type of example, if an application was developed in 1970, and the earliest possible relevant dates in the system could occur in 1970, then the system would be programmed to assign the century "20" to dates less than "70," and to assign the century "19" to dates greater than "70."

A "sliding" window allows the 100-year interval to advance according to a selected criterion. For example, the window can advance each year to better enable processing of "age" data. A "fixed" window uses a static 100-year period that does not change throughout the lifetime of the application.

The windowing techniques enable the conversion of applications at a potentially lower cost than database expansion. They also offer the potential of developing date modules that could be called from within applications to process dates. However, windowing techniques are not permanent and will provide future exposure to risk for long-lived applications. In addition, they will not effectively handle data that span more than 100 years (such as age). To avoid data integrity problems, there needs to be consistency throughout the organization on how dates are handled using the windowing approaches.

4.3.3 Field Compression

The compression approach packs 4-digit data into a 2-digit field. Compression enables the use of the current database and current fields in the database. However, program changes are still required to process the new format, and these changes will need to be applied consistently. In addition, conversion programs will need to be developed to apply the changed formatting. Once the changes are made, there may be data conversions at runtime, with potential operational performance issues.

4.3.4 Remediation Strategy Decision Factors

The correct option within an organization requires consideration of technical, strategic and business issues. It is important at this phase to perform a set of pilot remediations on a representative sample of the code in order to get realistic data points for the effort required by different correction strategies. A detailed plan for making the corrections will derive from the strategy. This plan may involve a hybrid of the database expansion, windowing, or compression strategies outlined above.

Decisions also need to be made on whether to make the corrections on screens and reports, and how to manage inter-operating converted and non-converted parts of systems. The planning decisions need to include approaches for the conversion of JCL and any support systems that interact with the application.

4.3.5 Compliance of Suppliers

Because most applications rely on supplier-developed operating systems, compilers, and support software, it is necessary to monitor the progress of all suppliers in their own remediation efforts. Four steps can be taken:

1. Compile a list of current supplier software, and relationship to all applications in an organization's inventory.
2. Track the supplier's stated timetable and progress to ensure that software can be Y2K compliant. A listing of a number of these efforts is kept by DISA [DISA 97].
3. Develop contingency plans and workarounds if suppliers do not meet critical schedules.
4. Test supplier software according to specific criteria. A checklist of issues for a number of different types of supplier software, including operating systems, compilers, local area networks and databases is provided by GTE [GTE 96].

4.4 Perform the Remediation

The recommended technical approach for making corrections is through automated transformations. The analysis of patterns and programming clichés can result in the development of a rule base for making the transformations automatically or semi-automatically. Some tools can assist in the transformations for specific narrow domains, such as the

programming languages COBOL or NATURAL. However, even in the most optimistic cases, automated transformations will cover substantially less than 100% of the cases. The rest of the corrections will then need to be made manually.

Some languages do not have automated tool support, and thus the entire set of transformations will need to be done manually unless further tools become available. The research agenda discussed below addresses areas of need for the development of additional tools.

4.5 Test the Changes

Testing is the task that is currently least understood. When 10% of the source code of an application is changed, testing is difficult, time-consuming and complex. With such a high percentage of changes to the source code, it is hard to demonstrate that the functionality is unchanged. A common estimate is that testing will require at least 50% of the effort for a Y2K project. Standard unit testing, integration testing, system testing and acceptance testing will be required for Y2K projects.

IBM [IBM 96] provides a framework for Y2K testing based on standard testing techniques. This approach includes the following:

- Unit testing and integration testing that are forms of structural testing whose primary goal is to locate errors created during remediation. These testing phases exercise all structures of the system. They include operations testing of normal production scenarios, stress testing of abnormal circumstances and volumes, and recovery testing after system failure.
- System testing and acceptance testing that are forms of functional testing whose primary goal is to locate design errors and improper implementation of requirements. The functional testing includes requirements testing, regression testing, error handling, manual support testing, and interface testing.

For most systems, it is hard to test whether the software works properly for future dates. Most databases do not contain dates past the year 2000. Injecting or simulating future dates is an option but not trivial to implement since most systems have to be modified in place. Moreover, verification of test runs with future dates have to be verified manually since there are only historical datasets available against which the results of the future data tests can be compared. IBM provides software packages for MVS that will intercept requests for the system date and substitute a future test date (e.g., February 29, 2000). Organizations that have change control processes, extensive regression test suites, and test harnesses in place should be able to manage the risk involved in Y2K testing.

Test scenarios need to be specifically developed for Y2K conversions. Simple data types like integers or dates are usually assumed to work properly and are not normally tested extensively. Guidelines are not currently available on test predictions or a minimal test suite.

IBM also offers some guidelines for test scenarios [IBM 96] The components to be considered include the following:

- Test processing cycles and functions that are activated on a regular basis. These include daily, weekly, monthly and annual cycles, as well as automatic archiving and restart functions.
- Test special dates, such as 12/31/99, 01/01/00, 02/29/00, and 03/01/00.
- Test time-sensitive data at critical dates.

There may or may not be enough excess computing resources to allow testing of all these systems (e.g., during the remaining weekends before the Y2K deadline). An additional complication to testing for the Y2K problem is that legacy systems often contain a number of different versions of compilers and software. As a result, it may be difficult to get a clean set of re-compiles and linkages. In the cases of older and potentially incompatible compilers and support software, it may be useful to attempt to do a full recompilation before testing begins to determine if a problem exists.

Little actual data is available on testing as applied to the Y2K problem since so few projects have actually completed or published the results of their Y2K conversions. This is an area that will require careful monitoring in the future.

4.6 Migrate to Production

To ensure continuous, safe, reliable, robust, and ready access to mission-critical functions and information, it will be necessary to migrate in place. The objective is to maintain the running system at all times and to perform the migration in small, incremental steps. The incremental approach will control risk. If a step fails, only that step will need to be repeated. The migration strategy needs to be developed incrementally to transition the corrected systems in a phased manner.

During migration, a number of tasks need to be performed, including

- development of procedures for conversion of data, either automatically or manually
- development of new programs for screens or reports that require the new date formats
- acquisition and installation of required storage devices and other hardware and software that will be needed for the conversion
- development of new job control programs and parameter files
- updating of all documentation
- backing up of old data files and load modules
- conducting parallel testing of new and old system components
- transferring of new components into a testing environment and placing into production on an incremental basis
- monitoring of system performance with each new increment

5. Technology Gaps

The discussion of the technical steps involved in addressing the Y2K problem leads to an identification of shortcomings in current technology ("technology gaps")—which adds to the challenge of tracking the Y2K problem. These gaps are discussed below in terms of the core technical tasks discussed above.

5.1 High-Level Systems Inventory

The development of a systems inventory is particularly difficult when a CM system is not used. While operating systems provide commands to facilitate the performance of the individual steps needed, there are no automatic and intelligent tools for a complete inventory analysis. A type of tool that could be particularly useful would be "crawlers" for specific operating systems. A "crawler" would analyze an entire operating system, identify all the executables, construct an inverse building map, and locate source files. The information could be gleaned by carefully investigating the file systems using version and library tags, scripts, makefiles, installation logs or README files, directory structures, and documentation. This type of tool could automate much of the systems inventory process. Without such a tool, manually creating a system inventory is an error-prone process.

5.2 Impact Analysis

A key to understanding code successfully and eventually to performing remediation on the code is parsing. While parser technology is relatively mature, it is somewhat uneven. Technology is needed to address the following needs:

- a standard intermediate language, such as that represented by Software Refinery, to serve as an interface between parser and impact-analysis tools, that is sufficiently detailed to handle Y2K analyses and conversions
- a grammar and a parser for every variant of a programming language
- parser generators for every grammar

After parsing the code, the analysis of date impacts can be aided through tools such as the following:

- a catalog of all known date formats and manipulations occurring in practice and their actual patterns in various languages
- pattern matching algorithms for locating date processing
- forward and backward impact analysis based on slicing technology
- assessment and evaluation of tools, algorithms, grammars, and catalogs

5.3 Remediation

To provide remediation for the Y2K problem, the following types of technology are needed:

- a catalog of algorithms for widening date formats in various languages
- a catalog of algorithms for windowing schemes in various languages
- a catalog of algorithms for date field compression in various languages

5.4 Test the Remediation

In the areas of inventory analysis, impact analysis, and remediation, tools have a significant impact on the Y2K problem. However, these phases only constitute about 30% of the overall life cycle of a Y2K migration project. One of the major goals should be automating the testing phase that typically contributes at least 50% toward the costs of a Y2K project. Test scenarios and coverage guidelines geared to Y2K projects are needed as projects begin to progress to the testing phase. Testing is especially important for mission-critical systems.

6. Requirements for a Coordinated Effort

Because of the time-critical urgency of the problem, it is important to develop a national program for the mobilization of resources. Many organizations are beginning to go through the discovery process. It will be important to provide lessons learned, templates, and transfer of knowledge in order to avoid re-inventing the same wheel. Coordination is required both at an organizational level and at a national level.

The components of a national agenda are derived through a combination of the technology gaps described in the previous section, as well as the management infrastructure and support that is required. This agenda needs to include

- development of awareness
- clearinghouses of tools, tool evaluations, and tool vendors
- research program for development of needed tools
- clearinghouse of software certification for Y2K compliance
- sharing of lessons learned and case studies
- general process templates

These components are each discussed briefly below.

6.1 Development of Awareness

One major theme is the difficulty in getting decision-makers to focus specific resources on addressing the Y2K problem [Martin 97, Kappelman 96, Cassell 97, Ragland 96]. It is important to disseminate information on the magnitude of the problem, its pervasiveness, the implications of failure to address the problem, and the fact that the solution to the problem is difficult.

6.2 Clearinghouses of Tools, Tool Evaluations, and Tool Vendors

A number of tools exist to address various aspects of the Y2K problem. Most of these apply to common languages for MIS problems, such as COBOL. However, there is little objective information on the functions or usefulness of tools. Frameworks are needed to categorize tools, and evaluations of their effectiveness and limitations are needed.

The Software Technology Support Center (STSC) has begun to address this problem in two ways: 1) an inventory of Y2K tools, and 2) an evaluation of several specific tools relevant for embedded systems (available in second quarter of 1997). In addition, MITRE has begun a tool clearinghouse, and disseminates information on its web page.

A well-funded effort to classify technology and tool functionality and to evaluate effectiveness is needed as new organizations launch Y2K projects and require guidance.

Numerous vendors have emerged to assist with the Y2K problem. Currently, lists of vendors are available from several web pages. However, the information about each vendor is derived from that vendor's publicity materials. It would be useful to develop a set of categories of specialization, by phase of lifecycle, languages and domains, as well as depth of experience and qualifications. Such a categorization would help potential customers to compare apples to apples and to make intelligent decisions.

6.3 Research Program for Development of Needed Tools

There is a need to mobilize and focus the research community. Tools are required in a wide number of areas, including the following:

- operating system crawlers
- parsers (by language)
- common program and intermediate representations
- transformation tools (by language and operating system)
- reusable algorithms for locating and correcting date problems
- static analysis tools
- testing tools
- regression test suite for date items

The first requirement is a careful inventory of what exists and where the gaps are. This would enable priorities to be set on the most significant gaps and it could guide research and policy decisions. A second requirement is the delivery and deployment of these research prototypes in an expedient manner.

6.4 Parser Clearinghouse

Because parser technology is central to understanding a system, one specific function of the research program could be the coordination of research on parser technology. While there are some clearinghouses of grammars for programming languages, there are no such facilities to pick up a parser for a particular language. Given a grammar for a particular programming language, a parser generator can be used to produce a parser automatically. Parser generators have been around for a number of years. However, the parsers produced by these generators provide output in their own format that is not usually compatible with impact analysis tools. A parser clearinghouse would provide a parser for a given variant of a programming language producing a common, language-independent intermediate representation that could then be used by language-independent impact analysis tools.

6.5 Technology Transition Issues

As important as the development of tools is, it will be equally important to pay attention to issues of usability and transition. Transitioning of technology from a research prototype to routine use within organizations is known to be difficult. In addition, it will be important to transition expertise across organizational boundaries. Although this type of transition is unprecedented and difficult, it is essential for solving the problem.

6.6 Clearinghouse of Software Certification for Y2K Compliance

As mentioned above, one aspect of the Y2K problem concerns the use of a large number of COTS software and complex interdependencies to applications software. It will be critical as the Y2K approaches to have an inventory of plans of vendors for Y2K compliance as well as certification to verify their compliance. DISA has begun such an effort and maintains a current list on its Web page.

6.7 General Process Templates

The overall process for addressing the Y2K problem is fairly common and universal. However, the firstfew organizations have spent a significant amount of time developing an overall process for solving the problem. MITRE-Tech has now developed generalized process templates [Bohner 96] that could be used by new organizations as initial guides for the process. These templates can be updated periodically as more organizations use them and provide feedback on them.

6.8 Sharing of Lessons Learned and Case Studies

Virtually every organization will need to address the Y2K issue for the majority of its systems. Since there is significant commonality in languages, domains, support systems, and operating systems, there can be common technical and management approaches and common lessons learned.

The sharing of lessons learned needs to be done both within organizations and between organizations. A structure similar to a Software Engineering Process Group (SEPG) can be created within an organization. This group would coordinate the sharing of tools, templates, processes and other reusable assets for solving the Y2K problem within an organization. It would also coordinate ideas and solutions with outside organizations to enable the greatest possible leveraging of common experiences in addressing a problem that is much larger than any single organization.

7. Conclusion

This report has summarized the overall issues and implications of the Y2K problem. The problem is pervasive and will affect practically all software in all organizations. Failure to address the problem can have severe consequences. We identified six core technical tasks required in addressing the problem. In our analysis we identified gaps in current technology, as well as the components of an overall strategy to address the issue.

Appendix A: Pointers to Resources

General Y2K web sites:

- SEI: http://www.sei.cmu.edu/~reengineering
- STSC: http://www.stsc.hill.af.mil/RENG/index.html#2000
- MITRE: http://www.mitre.org/research/y2k
- ISA: http://www.mitre.org/research/cots/COMPLIANCE_CAT.html
- IBM: http://www.software.ibm.com/year2000/
- BCS: http://www.bcs.org.uk/millen.htm
- GTE: http://www.mitre.org/research/cots/GTE_CRITERIA.html
- Peter de Jager: http://www.year2000.com

Parsing resource pages and research projects:

- Compiler construction tools: http://www.first.gmd.de/cogent/catalog
- Lexical analysis tools: http://www.cogs.susx.ac.uk/users/adrianh/lexical/
- C++ program analyzers: http://www.teleport.com/~smeyers/ddjpaper1.html
- LSME: http://www.cs.ubc.ca/spider/murphy/papers/lsme/tosem.html
- Software Refinery: http://www.reasoning.com

Slicing and impact analysis resource pages and research projects:

- NIST: http://www.ncsl.nist.gov/~jimmy/unravel.html
- U Wisconsin: http://www.cs.wisc.edu/wpis/html/
- Slicing Internet resources: http://www.unl.ac.uk/~mark/slicing.html
- Program slicing: http://www/cs.tu-bs.de/~krinke/Slicing/slicing.html

Appendix B: Interviews Conducted

A set of interviews was conducted in conjunction with the International Conference on Software Maintenance (ICSM) from November 4-8, 1996. This conference attracts researchers and practitioners in the filed of software maintenance. A number of Y2K talks and panels were conducted.

We selected the following three individuals to interview in depth:

- Shawn Bohner, MITRE-Tech
- Tom Driscoll, Formal Systems
- Mike Olsem, STSC

We interviewed these people because they had extensive experience in dealing with several organizations on the Y2K problem. We were able to capture insights on the problem from these knowledgeable individuals who had practical experience spanning a range of different organizations. Mike Olsem has been a leader in helping to coordinate the U.S. Air Force's efforts to resolve the Y2K problem. In that capacity he has evaluated a number of Y2K tools for the Air Force. Shawn Bohner has worked with a number of civilian U.S. government agencies and has developed general Y2K process templates. Tom Driscoll has developed a tool to analyze and remediate Y2K problems for the NATURAL language. Although this is a relatively small domain, he was able to provide general insights on the way in which corporations are dealing with the problem.

The interviews started with the individual's perspective of the major issues and problems concerning the Y2K issue. They then focused on the individual's involvement with Y2K efforts. We asked for their assessments of the current state of technology, gaps in the current technology, steps for addressing the problem by an organization, and recommendations for an overall strategy to deal with the problem.

References

Bohner 96 — Bohner, S. “Examining Year 2000 Data Challenges from the Maintenance Perspective.” *Proceedings, International Conference on Software Maintenance*. Monterey, Ca., November 4-8, 1996. Los Alamitos, Ca.: IEEE Computer Society Press, 1996.

Cassell 97 — Cassell, J.; Schick, K.; Hall, B.; & Phelps, J. “Management Edge: Year 2000 - Top View.” Stamford, Ct.: The Gartner Group,1997.

CGRO 96 — Committee on Government Reform and Oversight. “Year 2000 Computer Software Conversion: Summary of Oversight Findings and Recommendations.” U.S. Government Printing Office, Washington, DC, 1996.

de Jager 96 — de Jager, P. “Systematic Triage.” *American Programmer 9*, 2 (February 1996): 12-15.

DISA 97 — DISA. *DISA COTS Product Compliance Catalog* [online]. Available WWW: <URL: http://www.mitre.org/research/cots/COMPLIANCE_CAT.html> (1997).

GTE 96 — GTE. *Proposed Criteria for “Century Compliance”* [online]. Available WWW: <URL: http://www.mitre.org/research/cots/GTE_CRITERIA.html> (1996).

IBM 96 — IBM. “The Year 2000 and 2-Digit Dates: A Guide for Planning and Implementation.” IBM Report GC28-1251-00, Poughkeepsie, N.Y.: IBM Corp., 1996.

Kappelman 96 — Kappelman, Leon A., ed. *Solving the Year 2000 Computer Date Problem: A Guide and Resource Directory*. Chicago, Ill.: Society for Information Management (SIM) International, Year 2000 Working Group, 1996.

Martin 97 — Martin, R. “Dealing with Dates: Solutions for the Year 2000.” *Computer 30*, 3 (March 1997): 44-51.

Ragland 96 — Ragland, B. *Are You Ready for the 21st Century?* [online]. Available WWW: <URL: http://www.stsc.hill.af.mil/CrossTalk/1996/mar/AreYouRe.html> (1996).

Roberts 96 — Roberts, J. *Cost Estimation for Year 2000 Efforts* [online]. Available WWW: <URL: http://www.mitre.org/research/y2k/docs/BRIEF.html> (1996).

STSC — Software Technology Support Center. *Year 2000 Stand Alone Tools* [online]. Available WWW: <URL: http://www.stsc.hill.af.mil/RENG/y2ktools.html>.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (LEAVE BLANK)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 1997	Final

4. TITLE AND SUBTITLE
The Year 2000 Problem: Issues and Implications

5. FUNDING NUMBERS
C — F19628-95-C-0003

6. AUTHOR(S)
Dennis B. Smith
Hausi A. Müller
Scott R. Tilley

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

8. PERFORMING ORGANIZATION REPORT NUMBER
CMU/SEI-97-TR-002

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
HQ ESC/AXS
5 Eglin Street
Hanscom AFB, MA 01731-2116

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
ESC-TR-97-002

11. SUPPLEMENTARY NOTES

12.A DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified/Unlimited, DTIC, NTIS

12.B DISTRIBUTION CODE

13. ABSTRACT (MAXIMUM 200 WORDS)

A lot of attention has recently focused on the possibility that a great deal of software will fail at the turn of the century because of the way dates are stored and processed by computer programs. Attitudes range from alarmist to unconcerned regarding the magnitude and implications of the problem. This report outlines the basic issues of the so-called “Year 2000” (Y2K) problem and discusses some of its implications.

14. SUBJECT TERMS
legacy systems, reengineering, year 2000 (Y2K)

15. NUMBER OF PAGES
30 pp.

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL