Estimating Y2K Rework Requirements

Lee Fischman, *Galorath Incorporated*
Patricia A. McQuaid, *California Polytechnic State University*

Especially at this late juncture, critical decisions regarding year 2000 (Y2K) projects need to be grounded in an understanding of the true scope of rework requirements for assessed systems. This article discusses how to characterize Y2K renovation work and what the scope of that work might be.

As we all know, the Y2K problem is so serious that it may affect you personally. If you are being caught late in the game, engaged in an emergency bout of planning, the first thing you should ask yourself is, "What is the scope of my problem?"

Galorath Incorporated estimates resource requirements for software development, particularly the staffing and the scheduling required to accomplish a particular software project. Recently, we have been asked to estimate a number of Y2K renovation projects because of our traditional expertise in modeling software modification and reengineering; Y2K work is in many ways an extension of such traditional work. However, we have experienced a learning curve of our own. Although you can and should apply the lessons of the past, Y2K work does have a language and concerns of its own.

In this article, we share our experience estimating Y2K work using the SEER-Year2K model we have developed. Every Y2K job is different, so you first need to carefully assess your software inventory before applying any model. When estimating a Y2K job, you also should not apply rules of thumb developed on the basis of macro, even national-level data, if you want an accurate assessment of your costs.

There are many different recipes to describe Y2K activities; our research approach was to merge commonly accepted Y2K renovation activities with those we have developed to categorize other forms of rework.

Legacy Sizing

To obtain good Y2K renovation estimates, you must measure the size of your legacy code. The two most common metrics for software size are function points and lines of code, both of which are supported in our method. However, for Y2K work, lines of code are usually the preferred measure: It is far easier to apply an automatic line counter to legacy code than it is to develop a laborious, manual function point count. Furthermore, changes in software will usually be made in a line-by-line fashion whether by automatic or manual methods.

It is more difficult or impossible to develop meaningful line-of-code estimates for items like database files; for these situations, we rely on more ideally suited function points. A size estimate may thus combine lines of code with function points when necessary. As long as there is no overlap in what is counted, this is perfectly legitimate.

If you are able to size a good amount of your legacy code, but not all of it, you can apply this knowledge to areas you know less about. This is called *estimating by analogy*. Simple analogies are as easy as saying, "This and that have a similar magnitude." There are more sophisticated analogy procedures that can produce risk ranges, in addition to producing more accurate estimates.

Effective Size vs. Legacy Size

One big mistake made when scoping Y2K renovation requirements is to assume that *all* legacy software needs treatment. There is, in fact, a difference between total legacy size and the effective size of the software undergoing reparation. Our model computes effective size through a series of *rework percentages*, which is discussed in the next section. Imagine effective size as the contents of a box, as depicted in Figure 1.

Although total size represents all the code you own, effective size is the code impacted by Y2K rework requirements. These requirements may involve simple testing or actual changes. Changes may be made manually or using a Y2K "solution" product. Much of the effort required for a good Y2K rework estimate is therefore involved in assessing the effective size.

When doing an overall estimate of rework required, there is a balance between details required and essentially macro knowledge. We have learned that this dichotomy can be embodied in overall size estimates that are then adjusted downward using sets of specialized percentages; we call this the *adjustment to effective size*, a process that is by no means straightforward. You need to first define the percentage items, then figure out how these percentages should be used to adjust the gross size estimate. We have acquired many data sets of projects completed, in addition to much heuristic knowledge and post-validations of our estimates, which have guided us toward proper definition of percentages and their formulaic specification.

Rework Items

This section describes rework categories for Y2K renovation that we have developed for our estimates. These originate from our knowledge of renovation issues, the experience of others, and our recent Y2K consulting engagements. Although the computations used are

Figure 1. *Effective size of the software undergoing remediation.*

specific to our model, the definitions should provide robust ground rules to evaluate your work. We have added detailed "Y2K Advice" sections to guide you in this. The rework items covered include

- Reverse Engineering Required.
- Date-Related Design Change.
- Specification Updates Required.
- Manual Recoding Required.
- Automated Recoding Required.
- Automated Conversion Verification.
- Programmer and Unit Testing Required.
- Test-Bed Preparation.
- Application Testing.

The diversity of these categories is a strength, since misspecification of any one category is not likely to drive the estimate too far off. The following sections provide details on these rework categories.

Reverse Engineering Required

This is the percentage of code (relative to the total application size) that a technical staff must review to understand what is happening at the code level. Include only those lines of code with which someone must be familiar for the application to be considered "reviewed."

Whole blocks of code may theoretically undergo "review," but lines thoroughly analyzed may be slight. In this case, the percentage of reverse engineering would only be the lines studied. Let us say that a "100 percent code review" translates into a scan across all code but only at the level of function calls—function contents are not examined. In this case, the percentage of reverse engineering is far lower than 100 percent—it may even be 1 percent or less.

To determine the percentage of reverse engineering, consider

- The level of familiarity with the system's internal logic. Systems that have been informally maintained over the years may now require basic understanding (ultimately expressed as flowcharts, entity-relationship diagrams, data flow diagrams, data dictionaries) before substantial renovation can begin.
- Formal documentation requirements may mandate additional reverse engineering, other than that necessary to complete work.
- Redocumenting requirements. These may be closely related to reverse engineering.

Y2K Advice

Outside teams may have to reverse engineer code to orient themselves to basic architecture and design. Code analyzers, scanning methods, and other automated tools may mitigate the need to reverse engineer—do not include the code covered by automated methods in this percentage. Older applications have higher levels of hidden utility and therefore require a more detailed approach, which often translates into additional reverse engineering.

Date-Related Design Change

These are date-related design changes measured relative to the total application size. Design is at a level that covers everything except actual programming.

To determine the percentage of redesign required, consider changes to

- Data structures, e.g., data-type changes, new or deleted fields, and expansions. When redesigning these, think in terms of the amount of code necessary to carry the design in a given data structure (such as in a structured query language CREATE command).
- Object methods.
- Date-related data passage between functions.
- Operating system-related issues, e.g., memory usage and date and time functions.
- Design changes at the function level (this does not include isolated code changes that do not change the function's design).

Y2K Advice

Consider the changes that actively address date issues. These include bridges to noncompliant external applications, encapsulation of potentially troublesome code to capture noncompliant dates, on-the-fly record format translation, windowing of two-digit years, redesign of date logic, representation, and manipulation.

Specification Updates Required

This is the percentage of new documentation to be developed compared to total existing documentation.

Redocumentation relates to both the proportion of a system being redocumented and the coverage of that documentation. An application may be well described at a high level, but this may only amount to a small percentage of what comprises the application. Only if requirements spell out that every single line of code be mentioned in new documentation is redocumentation 100 percent.

Redocumentation can also be thought of as the amount of code of which a technician must be cognizant to adequately document a system; in this way, it may be closely related to reverse engineering. Thus, a short report that describes a large system, which took only a few days to produce, is likely to fall into the low percentage ranges.

To determine the percentage of updates required, consider

- Comments. When inserted into the code, they are not part of formal documentation; however, this may be a part of reverse engineering.
- Documentation. If documentation must be completely rewritten, this does not necessarily mean 100 percent redocumentation. Documentation is calculated only with respect to the full application.
- Formal documentation requirements. Informal development shops with no formal processes or standards have documentation requirements that are typically orders of magnitude lower than those for shops that follow stringent standards.

Y2K Advice

Consider the state of existing formal documentation for this application, then decide whether it has to be updated. If there is no outstanding formal documentation requirement (have previous maintenance efforts had any?), the updates required percentage may be zero. An outsourcer may have formal documentation requirements imposed to ease later in-house maintenance or

because of other contractual requirements.

Manual Recoding Required

These are manual changes to software evaluated as a percentage of existing size. Thus, if 200 lines of a 10,000-line application are modified, recoding would be 2 percent.

To determine the percentage of recoding, consider

- New code. If any new code is being written to support changes, it should be factored into the size of the existing software to develop a correct percentage. Thus, if 200 lines of a 10,000-line application are modified and 200 new lines are written, recoding will be 4 percent.
- Language conversions. Major language changes, such as from COBOL to C (with no automatic conversion aids), will require 100 percent recoding, even if virtually no redesign is required.
- Minor changes due to a change in compilers.

Do not count code that is changed using an automated tool, but do consider the manual refinements that are necessary after the tool is used.

Y2K Advice

Consider code that directly impacts date and time issues. This includes, to the extent applicable, data typing and initialization of date and time variables, date logic, input and output of date and time data, encapsulation of existing code, bridges to noncompliant external applications, and other software patches necessary to ensure compliance, fault recovery, etc. A simple text scan helps reveal what percentage of the code needs attention and rewriting.

Automated Recoding Required

These are automated changes to software evaluated as a percentage of the existing size. Thus, if 500 lines of a 10,000-line application are to be modified by means of an automated tool, automated recoding would be 5 percent.

To determine the percentage of automated recoding, consider

- If any new code must be written to support the automated changes, it should be factored into the size of the existing software to develop a correct percentage. Thus, if 500 lines of a 10,000-line application are to be recoded by an automated tool, but 100 new lines must first be written to assist the automated recoding, automated recoding will be 6 percent, and manual recoding must be increased by 1 percent.
- If no automated tool is used, this input will be 0 percent.

Automated Conversion Verification

This is the amount of code processed by automated conversion that requires manual inspection, review, or walkthroughs. This should be given as a percentage of the code that is being converted by automated means. Code reviews are generally done to ensure coding standards and conventions are adhered to and to detect potential errors.

To determine the percentage of verification required, consider

- Code reviews are work that is subject to low-level, direct review by one or more people knowledgeable in the organization's standards and practices and the language in which the code is written.
- The use of automated code conversion may significantly reduce the need for code reviews or at least reduce the rigor of required reviews.
- If no formalized code reviews are performed, i.e., regular get-togethers, this could be 0 percent.
- Use of automated tools to check for adherence to coding standards and conventions may reduce or eliminate code reviews.

Y2K Advice

Are code reviews part of your normal development or maintenance process? If so, they will probably be part of your Y2K renovation. Code reviews are normally isolated to new changes. To develop an accurate percentage, consider the percentage of work that is normally reviewed and multiply this by the percentage of new coding.

Programmer and Unit Testing Required

This is measured as the percentage of code, relative to existing size, that requires unit testing. Unit testing is generally done by programmers to test and debug low-level software. Unit testing is usually considered at the module level (such as functions and subroutines) and the unit level (generally a source file).

To determine the percentage of unit testing, consider

- The amount of recoding required, because code changes typically must be tested. If 10 percent of code is changed and all of that is tested, unit testing is also 10 percent.
- External testing. If testing is carried out by people other than the programmer making code changes, these testers will probably cover more code than has been changed.

Y2K Advice

Are unit tests part of your normal development or maintenance work? If so, they will probably be part of your Y2K renovation. In addition, because unit tests of certain sensitive functions are an efficient way to track down faults, unit testing may exceed the amount of code changed.

Test-Bed Preparation

This parameter covers the preparation of new test plans and test procedures, not their implementation. Actual testing is covered in application testing. Test plans and procedures that already exist should not be included in the Test-Bed Preparation percentage.

Test preparation describes the scope, approach, resources, and schedule of test activities. It further identifies test items, features to be tested, tasks, who will perform each task, and any risks that require contingency planning. To clarify the difference between test procedures and plans, imagine an orchestra: Test plans describe the conductor's job, whereas test procedures describe the musician's instructions.

The percentage of test-bed preparation required relates to both the proportion of a system to be tested and the depth of testing required. In an

absolute sense, the overall percentage covers the extent to which the lowest logical attributes of the software are exercised. If plans spell out that every piece of code be reached by test plans and procedures, the percentage required is 100 percent.

To determine the percentage of test-bed preparation required, consider

- If you have informal integration testing and little or no formal testing, test-bed percentages are likely to be low.
- Test plans orchestrate testing activities but do not control the most detailed tasks. These are covered by test procedures.
- Can existing test plans be reused without modification? For every test plan that exists, somewhat fewer new plans may be required. The same may be true for existing test procedures.
- A clever test plan may simultaneously exercise multiple test points with a single directive. For instance, a repetitive software architecture may allow the test plan to specify an identical approach across system components. If each "test point" is separately accounted for in development of the test plan, coverage should include each test point separately. However, if only a single test point needs to be accounted for despite several being tested, coverage should include only that test point.

Y2K Advice

A Y2K renovation effort may require test plans and procedures if formal testing or third-party regression and integration testing is used. Furthermore, validation of date compliance may require that additional tests be drawn up.

Application Testing

This parameter covers only actual testing, not detailed test preparation. Preparation is covered in Test-Bed Preparation. Include all testing effort, even if familiar from previous efforts. Formal tests are conducted in an environment of intentional yet usually amicable mistrust; an outside authority or an in-house authority—which requires extremely formal turnover procedures—asks developers to provide proof that various aspects of a system work before they will accept delivery. Formal testing is sometimes called user acceptance testing.

The overall formal testing percentage covers the extent to which the lowest logical attributes of the software are exercised. Some formal tests provide an automatic environment that is designed to ultimately "contact" a high percentage of test points. If so, the percentage of formal tests required should be rated *lower* than the percentage of points that will be asymptotically "hit." The test percentage should instead be rated at the percentage of the application that the equivalent test effort could at *minimum* cover.

To determine the percentage of application testing required, consider:

- Do you do formal testing? If it is not a part of the standard product sign-off, formal testing will be zero.
- Complete formal testing does not necessarily imply "100 percent"—what percentage of code is actually exercised?

Y2K Advice

Meaningful formal acceptance testing is common in so-called "formal" development environments.

Rework Percentages

Recall that all the categories above are expressed in terms of percentages. The first "default" percentages that we developed were based on our experience with other types of renovation work; these can be modified by a user possessing more specific information. As a sanity check, however, it is useful to note that with these default percentages, our model yields results that are similar in magnitude to those produced by other third-party benchmarks, notably those by the Gartner Group and Capers Jones.

In Table 1, "Least" is the least likely coverage or percentage, "Likely" is most probable, and "Most" is the highest possible. These percentages are translated into effort and schedule estimates via SEER-Year2K's analytic model.

The percentages are in some ways quite general; what is important are

- The magnitudes being assumed.
- The balancing for the ranges specified.

It also is apparent that the percentages are quite low, emphasizing how strictly we have defined activities. This should make sense, because Y2K renovation falls far short of rewriting code. We

Table 1. *Rework percentages. Some are in hundredths because they are products of other estimating formulas.*

	Automatic			Semiautomatic			Manual		
	Least	Likely	Most	Least	Likely	Most	Least	Likely	Most
Reverse Engineering Required	0.00	1.00	3.00	0.00	2.00	6.00	0.00	4.00	12.00
Date-Related Design Change	0.00	4.00	4.00	0.00	4.00	4.00	0.00	4.00	4.00
Specification Updates Required	0.00	0.00	2.00	0.00	0.00	2.00	0.00	0.00	2.00
Manual Recoding Required	0.00	0.00	0.00	0.00	2.00	2.00	0.00	4.00	4.00
Automated Recoding Required	0.00	4.00	4.00	0.00	2.00	2.00	0.00	0.00	0.00
Automated Conversion Verification	1.00	5.00	10.00	1.00	5.00	10.00	0.00	0.00	0.00
Programmer and Unit Testing Required	0.00	1.14	1.30	0.00	2.58	2.65	0.00	4.00	4.00
Test-Bed Preparation	0.00	0.16	0.68	0.00	0.16	0.68	0.00	0.16	0.68
Application Testing	0.00	4.00	4.00	0.00	4.00	4.00	0.00	4.00	4.00

reiterate that these percentages are starting points; they need to be plugged into a formula to determine cost and—banish the thought with our millenium deadline—schedule. Over time and with enough projects, you could, as we have, develop such a formula—maybe by the millenium?

Conclusion

A diversity of activities are under the Y2K umbrella, and not all are always required. Sizing alone, therefore, cannot suffice as an accurate guide to effort required. It is useful to develop a set of secondary criteria based on types of activity.

You will need a standard way to specify the magnitude of Y2K rework activities; we have found that percentages are an efficient standard metric to scope activities. Percentages are well-suited to the gross inventorying that typically occurs in Y2K planning. Application specialists in the field with day-to-day responsibility for Y2K-impacted systems and the renovation team doing the work will buy in to the language of percentages with a minimum of introduction. Percentages thus give you an agreeable basis to rapidly approach an estimate.

The importance of a risk-based estimate cannot be overstated given the scarce resources and tight schedules of Y2K work. Without knowing upside or downside exposure, a simple point estimate in the face of such constraints carries tremendous risk. With a deadline that cannot be moved for these renovations, risk is not an option. ◆

About the Authors

Lee Fischman is special projects manager at Galorath Incorporated in El Segundo, Calif. He is active in the development of SEER tools and consulting methods. He wrote the Software Evaluation Guide for the Office of the Secretary of Defense, Program Analysis and Evaluation, and he has explored software economics and estimating in numerous papers over the past several years, all available at http://www.galorath.com.

Galorath Incorporated
Voice: 310-414-3222
E-mail: fischman@galorath.com
Internet: http://www.galorath.com

Patricia A. McQuaid is an assistant professor of management information systems at California Polytechnic State University at San Luis Obispo. She has taught a wide range of courses in both the business and the engineering colleges. She has industry experience in computer auditing and is a certified information systems auditor. Her research interests include software process improvement, software quality, and software testing, particularly in complexity metrics. She is the developer of a new software complexity metric known as the Profile Metric.

California Polytechnic State University
Management Information Systems Area
College of Business
San Luis Obispo, CA 93407
Voice: 805-756-5381
Fax: 805-756-1473
E-mail: pmcquaid@calpoly.edu

Coming Events

Configuration Management Seminars

Dates: Between Jan. 25, 1999 and March 5, 1999, depending on location and seminar.
Locations: San Diego, Calif., Las Vegas, Nev., Orlando, Fla., Washington, D.C.
Sponsor: Technology Training Corporation
Instructors: Robert Ventimiglia, Larry Bowen
Topics: Four seminars. Examples of topics: How to Integrate Configuration Management (CM) into Your Software Development Methods, the Latest CM Standards and Requirements, Establishing Appropriate Baselines.
Contact: Dana Marcus
Voice: 310-563-1223
E-mail: dmarcus@ttcus.com

ITCC World Conference and Exposition

Dates: Feb. 11-12, 1999
Location: Chicago, Ill.
Topic: Information Technology (IT) Consultants and Contractors (ITCC) World Conference and Exposition is the only industry event designed specifically for IT consultants and contractors. Join over 60 of the region's top IT consulting and software companies at the ITCC Exposition, along with a two-day conference program and workshops to maximize your professional success.
Internet: http://www.itccexpo.com

ESC Spring: Embedded Systems Conference

Dates: March 1-4, 1999
Place: McCormick Place South, Chicago, Ill.
Topic: Five days of high-level technical training as well as a three-day product exhibition. Target audience: embedded systems software developers, hardware engineers, and project leaders.
Internet: http://www.embedded.com/escfrm.htm

SEPG 99: 11th Software Engineering Process Group Conference

Dates: March 8-11, 1999
Location: Atlanta, Ga.
Subject: This four-day event brings together international representatives from government, industry, and academia for a global perspective on software process improvement.
Sponsor: Software Engineering Institute
Voice: 412-268-3007
Fax: 412-268-5758
E-mail: sepg@sei.cmu.edu