



Determining the Distribution of Maintenance Categories: Survey versus Measurement

STEPHEN R. SCHACH

srs@vuse.vanderbilt.edu

Department of Electrical Engineering and Computer Science, Vanderbilt University, Nashville, TN 37235, USA

BO JIN

bo.jin@vanderbilt.edu

Department of Electrical Engineering and Computer Science, Vanderbilt University, Nashville, TN 37235, USA

LIGUO YU

liguo.yu@vanderbilt.edu

Department of Electrical Engineering and Computer Science, Vanderbilt University, Nashville, TN 37235, USA

GILLIAN Z. HELLER

gheller@efs.mq.edu.au

Department of Statistics, Macquarie University, Sydney, NSW 2109, Australia

JEFF OFFUTT

ofut@ise.gmu.edu

Department of Information and Software Engineering, George Mason University, Fairfax, VA 22030, USA

Editor: Marvin Zelkowitz

Abstract. In 1978, Lientz, Swanson, and Tompkins published the results of a survey on software maintenance. They found that 17.4% of maintenance effort was categorized as corrective in nature, 18.2% as adaptive, 60.3% as perfective, and 4.1% was categorized as other. We refer to this as the “LST” result. We contrast this survey-based result with our empirical results from the analysis of data for the repeated maintenance of three software products: a commercial real-time product, the Linux kernel, and GCC. For all three products and at both levels of granularity we considered, our observed distributions of maintenance categories were statistically very highly significantly different from LST. In particular, corrective maintenance was always more than twice the LST value. For the summed data, the percentage of corrective maintenance was more than three times the LST value. We suggest various explanations for the observed differences, including inaccuracies on the part of the maintenance managers who responded to the LST survey.

Keywords: Maintenance categories, open-source software, repeated maintenance, real-time product, Linux, GCC.

1. Introduction

One of the most widely cited papers on software maintenance is “Characteristics of Application Software Maintenance” (Lientz et al., 1978). The authors of that paper analyzed 69 responses from maintenance managers to a 35-page questionnaire containing 50 different questions. Even though the survey was conducted 25 years

ago, one result from that survey continues to be quoted regularly, namely, the relative frequency of adaptive, corrective, and perfective maintenance. For example, the latest editions of all three top-selling textbooks in software engineering quote this result (Schach, 2002; Pressman, 2001; Sommerville, 2001).

In more detail, Lientz et al. (1978) stated that 17.4% of maintenance effort was categorized as corrective in nature (“emergency fixes, routine debugging”); 18.2% as adaptive (“accommodation of changes to data inputs and files and to hardware and system software”); 60.3% as perfective (“user enhancements, improved documentation, recoding for computational efficiency”); and 4.1% was categorized as “other” (Lientz et al., 1978). For brevity in what follows, we will refer to this as the LST result.

Although these results are valuable and useful, they are somewhat dated. Since the paper’s 1978 publication date, there have been considerable changes in the IT industry. We use new technologies, new processes and procedures to design and develop software, and there are numerous new types of applications. A significant change is that software now heavily relies on reuse, which can directly impact maintenance. Thus, we have decided to revisit the categorization of maintenance changes.

We have recently compiled detailed data for the repeated maintenance of three software products: RTP, a widely used commercial real-time product (Wang et al., 2001); Linux, the open-source operating system (Linux Online, 2000); and GCC, the open-source set of compilers (GCC Home Page, 2001). One of the items we measured was the distribution of maintenance categories. We fully expected that our results would be in accordance with LST. Much to our surprise, the distributions we observed were vastly different from LST.

We therefore felt that it would be appropriate to carefully reexamine LST. For example, suppose that a software organization is devoting 50% of its maintenance effort to corrective maintenance. If LST is correct then the number of faults in the software developed by that organization is unacceptably high. But if the results presented in this paper can be validated for software as a whole, then the number of faults is slightly below average. LST is widely accepted, so correcting LST could have major implications for the management of maintenance.

Previous work in this area includes (Nosek and Palvia, 1990), in which results similar to LST were obtained when the same questionnaire was sent out 10 years later. Results of automatic categorization of maintenance performed on a real-time software system are described in Mockus and Votta (2000). Unfortunately, the maintenance categorization used in that paper is inconsistent with other papers, so the results are hard to compare. For example, Mockus and Votta (2000) use the term “adaptive maintenance” to mean “adding new features.” They also introduce additional categories, such as “inspection maintenance” (the result of a code inspection).

In Section 2, we discuss possible granularities for measuring maintenance categories, and in Section 3, we describe the granularities we used in this study. Sections 4, 5, and 6 contain our maintenance data for RTP, Linux, and GCC, respectively. In Section 7, we discuss LST. Our conclusions are in Section 8.

2. Granularity of Maintenance Category Data Measurement

Suppose we wish to categorize maintenance activities as adaptive, corrective, perfective, or other. This categorization can be performed on the basis of measurements at various levels of granularity, including the line of code level, change-log level, module level, and program level. At the end of this section, we contrast measurements made at these levels of granularity with the approach used in determining LST.

2.1. Line of Code Level

Using a utility like *diff*, each line that was changed (inserted, modified, or deleted) in the course of producing a new version of the program is flagged. Then, the category of the change to that line is determined by examining the change and deciding whether it is adaptive, corrective, perfective, or other (the categorization used in LST). This level of categorization clearly provides maximal information regarding the nature of the maintenance performed. A disadvantage is that we have found that gathering data at this level of granularity is exceedingly time consuming. Also, it can be hard to analyze such changes statistically when the number of lines of code decreases from one version to the next, especially when the decrease is large. For example, version 2.3.31 of Linux kernel module **Sched.c** has 2090 lines of code, whereas version 2.3.32 has only 1420 lines of code, a 32% decrease.

2.2. Change-Log Level

Typically, a change-log text file consists of entries like “warn the user that all but four cases have been disabled” or “prevent an endless loop when -1 is stored in the hash table.” Implementing these changes could require the addition, deletion, and modification of several lines of code in several different modules. At the change-log level, all these changes to the code are considered to be one unit of maintenance and are recorded on that basis. A strength of this approach is that it reflects the maintenance programmers’ view of the different activities that were performed. The major weakness is that it does not distinguish between correcting just one line of one comment within one module (which we have actually observed), and completely rewriting many modules to correct one fault (which we have not yet observed). Both maintenance operations would be recorded as one corrective change-log level modification.

2.3. Module Level

At the change-log level, as explained in the previous section, each entry in the change log constitutes one unit of maintenance. At the module level, we treat all the changes

made to a module as a single unit of maintenance. If all the changes made to a specific module are (say) adaptive, then we classify that unit of maintenance as adaptive. However, if at least one change is (say) corrective, then we classify the maintenance as adaptive/corrective. Then, for the purpose of statistical analysis, the set of changes to that module are deemed to be half adaptive and half corrective. A disadvantage is that this may not be an accurate measure of the relative effort when maintenance of more than one category is performed on a module.

2.4. Program Level

Here we consider the program as a whole and treat all the changes made from one version to the next as one unit of maintenance. As with module level categorization, we then categorize the change to a program as (say) adaptive or (say) adaptive/perfective. The strength of this approach is that we get the “big picture.” The major weakness is that the resulting data do not indicate the scale of the change, for example, how many modules were changed, let alone the extent of the changes to each module. As with module level categorization, there is also the problem of how to treat (say) adaptive/corrective/perfective maintenance accurately; the assumption that equal effort was devoted to adaptive, corrective, and perfective maintenance may not be a fair reflection of what was actually done.

2.5. Approach used to Determine LST

As explained in Section 6, Lientz et al. (1978) asked maintenance managers to estimate the relative percentage of effort devoted to specific maintenance activities for a specific application software system. That is, no measurements as such were performed. Instead, managers estimated how much effort was devoted to each of the activities, and then stated how confident they felt about their estimate. The activities were then grouped into four categories: corrective, adaptive, perfective, and other (Swanson, 1976). For example, the two activities “emergency fixes” and “routine debugging” were grouped into corrective maintenance.

In order to ensure that our results were comparable with those of LST, in our study we used the same four categories as LST.

3. Methodology

For RTP (Section 4), we determined the maintenance categories at only the module level. In the case of Linux and GCC (Sections 5 and 6), we determined the maintenance categories at both the module level and the change-log level in order to obtain two different views of repeated maintenance. In this paper, we report on all the distribution data we obtained at both the module level and the change-log level.

Table 1. Threshold values for Cohen's Kappa statistic (El Emam, 1998).

Kappa value	Strength of agreement
< 0.45	Poor (bottom 25%)
0.45–0.62	Moderate (bottom 50%)
0.63–0.78	Substantial (top 50%)
> 0.78	Excellent (top 25%)

For many software products, there is no change log per se. In such cases, entries similar to those of change logs are sometimes found as comments in the code; this is how changes to RTP are notated. When neither a change log nor comments were available (as, for example, with much of Linux), we used *diff* to find what changes had been made and then constructed a change log on the basis of the changes to the code.

At both the change-log level and the module level, we categorized each change using the LST categories. This approach is potentially subjective, so we wanted to make the categorization as objective as possible. Accordingly, we performed each classification twice and computed the cross-rater reliability. Two researchers independently categorized each change. Their evaluations were then compared using Cohen's Kappa test (Cohen, 1960).

Cohen's Kappa coefficient κ (Cohen, 1960) is an index of cross-rater reliability. That is, κ is a measure of the extent to which two raters agree. Let P_o denote the proportion of ratings on which the two raters agree, and P_e denote extent of agreement expected by chance, that is, the expected proportion of ratings that agree, assuming the raters are statistically independent. Then κ is defined as

$$\kappa = \frac{P_o - P_e}{1 - P_e}$$

That is, κ is the ratio of the observed excess over chance agreement to the maximum possible excess over chance agreement. We used the Kappa threshold values determined by El Emam (1998); these values appear in Table 1.

We now present our results on the repeated maintenance of RTP, Linux, and GCC, in each case identifying the level of granularity of the maintenance data that we extracted, and indicating the level of agreement regarding the categorization.

4. Repeated Maintenance of RTP

We have analyzed the repeated maintenance of RTP, a widely used PC-based commercial real-time product written in a combination of Assembler and C. The size of the product is about 12 KLOC. Procedures are grouped into 10 files; seven of the files consist of Assembler procedures and the other three contain C functions. We were provided with 148 versions of those 10 files, that is, the 10 original versions plus 138 modified versions created between 1987 and 1996 (for reasons of trade secrecy,

Table 2. Data at the module level for the 138 changed modules of RTP.

Maintenance category	Observed number	Observed percentages (%)	Expected percentages (LST) (%)
Adaptive	19	13.8	18.2
Corrective	59	42.8	17.4
Perfective	37	26.8	60.3
Other	23	16.7	4.1
Chi-square test	$P < 0.001$		

Table 3. The cross-rater reliability analysis for RTP at the module level.

Rater 2	Rater 1				Total
	Adaptive	Corrective	Perfective	Other	
Adaptive	17	0	0	0	17
Corrective	2	58	4	2	66
Perfective	0	1	32	2	35
Other	0	0	1	19	20
Total	19	59	37	23	138

Table 4. The value of the Kappa coefficient of reliability for each categorization.

	RTP	Linux	GCC
Module level	First set of versions	0.81	0.81
	Middle set of versions	0.92	0.82
	Last set of versions	0.70	0.82
Change-log level	First set of versions	0.81	0.80
	Middle set of versions	0.81	0.86
	Last set of versions	0.74	0.89

we were not given access to the latest versions). Our complete results may be found in a companion paper (Wang et al., 2001). Table 2 contains module level data for the 138 modified versions.

The chi-square test (Weiss, 1995) was used to compare the observed distribution of maintenance categories with the distribution expected according to LST. As can be seen from Table 3, the probability that the distribution of maintenance categories we observed was drawn from a population distributed according to LST is < 0.001 . That is, statistically the distribution of maintenance categories that we have observed is very highly significantly different from that in LST.

As stated in Section 3, two different evaluators categorized the changes separately. The cross-rater reliability analysis is presented in Table 3. The second column of Table 4 shows that the value of the Kappa coefficient is 0.87. From Table 1, this means that the strength of the agreement is considered excellent.

5. Repeated Maintenance of the Linux Kernel

We then examined 391 versions of Linux, from version 1.0 through version 2.3.51 (Schach et al. 2001). We concentrated our efforts on the Linux kernel because there are only 17 kernel modules and 6506 versions of those modules; in contrast, the current version of Linux has nearly 2000 modules, and there are up to 390 previous versions of each of those modules. In other words, our Linux maintenance research project was manageable because we restricted our efforts to measuring various aspects of “only” 6506 modules.

Here we report on both module level and change-log level data, as explained in Section 3. (As stated in Section 3, we constructed a change log from the changes to the source code.) We were particularly interested in determining whether the maintenance phase can be divided into subphases, so we considered the first 20 versions, the middle 20 versions, and the last 20 versions of the 391 versions of Linux at our disposal. The result of comparing each set of 20 versions against the LST distribution at the module level data is shown in Table 5, and at the change-log level data in Table 6. (In Table 5, two of the “observed numbers” are fractions. This is because the 15 instances of corrective/perfective maintenance at the module level are treated as 7.5 instances of corrective maintenance and 7.5 of perfective maintenance, as explained in Section 2.3.)

Chi-square tests were again used to compare the observed distributions of maintenance categories with the distribution expected according to LST. As can be

Table 5. Data at the module level for the first 20, middle 20, and last 20 versions of the Linux kernel.

Maintenance category	First 20 versions		Middle 20 versions		Last 20 versions		Expected percentages (LST) (%)
	Observed number	Observed percentages (%)	Observed number	Observed percentages (%)	Observed number	Observed percentages (%)	
Adaptive	2	2.2	0	0.0	0	0.0	18.2
Corrective	48	53.3	42	73.7	30.5	50.8	17.4
Perfective	34	37.8	11	19.3	25.5	42.5	60.3
Other	6	6.7	4	7.0	4	6.7	4.1
Chi-square test	$P < 0.001$		$P < 0.001$		$P < 0.001$		

Table 6. Data at the change-log level for the first 20, middle 20, and last 20 versions of the Linux kernel.

Maintenance category	First 20 versions		Middle 20 versions		Last 20 versions		Expected percentages (LST) (%)
	Observed number	Observed percentages (%)	Observed number	Observed percentages (%)	Observed number	Observed percentages (%)	
Adaptive	2	0.9	0	0.0	0	0.0	18.2
Corrective	87	40.5	151	78.6	75	53.6	17.4
Perfective	115	53.5	37	19.3	61	43.6	60.3
Other	11	5.1	4	2.1	4	2.9	4.1
Chi-square test	$P < 0.001$		$P < 0.001$		$P < 0.001$		

Table 7. The cross-rater reliability analysis for the first 20 versions of the Linux kernel at the change-log level.

Rater 2	Rater 1				Total
	Adaptive	Corrective	Perfective	Other	
Adaptive	2	0	0	0	2
Corrective	0	82	16	0	98
Perfective	0	5	99	2	106
Other	0	0	0	9	9
Total	2	87	115	11	215

seen from Tables 5 and 6, in all cases the probability that an observed distribution came from a population with the LST distribution is < 0.001 . We again deduce that the observed distribution of maintenance categories is statistically very highly significantly different from the LST distribution.

Again, two different evaluators categorized the changes separately. For the sake of space, only the cross-rater reliability analysis for the first 20 versions of the Linux kernel at the change-log level is presented here (Table 7). The values of all six Kappa coefficients appear in the fourth column of Table 4; four of the six Kappa coefficients were 0.80 or higher so, from Table 1, the strength of the agreement is considered excellent. The other two coefficients were 0.74 and 0.70; here, the strength of the agreement is considered substantial.

6. Repeated Maintenance of GCC

Next, we examined versions 2.4.0 through 2.7.2.3 of GCC (GNU Compiler Collection), a set of open-source compilers for C, C++, Fortran, Objective C, and other languages, published by the Free Software Foundation (GCC Home Page, 2001). The current version of the source code consists of over 1000 modules totaling nearly 850,000 lines of code. Just under 200 of the modules are procedural C code (.c) and just under 500 are C header modules (.h).

Again, we report on both module level and change-log level data, for the reason given in Section 3. Also, as with Linux, we wanted to determine whether or not the maintenance phase can be divided into subphases, so we considered the first five versions, the middle five versions, and the last five versions of GCC. Unlike the Linux data, in the case of GCC we did have access to a change log.

The module level data are shown in Table 8, and the change-log level data in Table 9. Once again using chi-square tests, we deduce that, as in the case of Linux and RTP, the observed distribution of maintenance categories is statistically very highly significantly different from that expected according to LST.

Table 8. Data at the module level for the first five, middle five, and last five versions of GCC.

Maintenance category	First five versions		Middle five versions		Last five versions		Expected percentages (LST) (%)
	Observed number	Observed percentages (%)	Observed number	Observed percentages (%)	Observed number	Observed percentages (%)	
Adaptive	1.833	1.0	3.5	3.3	4.333	6.3	18.2
Corrective	87.833	50.2	56.5	53.8	46.833	67.9	17.4
Perfective	85.333	48.8	42	40.0	17.833	25.8	60.3
Other	0	0.0	3	2.9	0	0.0	4.1
Chi-square test	$P < 0.001$		$P < 0.001$		$P < 0.001$		

Table 9. Data at the change-log level for the first five, middle five, and last five versions of GCC.

Maintenance category	First five versions		Middle five versions		Last five versions		Expected percentages (LST) (%)
	Observed number	Observed percentages (%)	Observed number	Observed percentages (%)	Observed number	Observed percentages (%)	
Adaptive	3	1.0	8	3.9	13	11.8	18.2
Corrective	155	51.5	117	57.1	74	67.3	17.4
Perfective	143	47.5	75	36.6	23	20.9	60.3
Other	0	0.0	5	2.4	0	0.0	4.1
Chi-square test	$P < 0.001$		$P < 0.001$		$P < 0.001$		

Table 10. The cross-rater reliability analysis for the first five versions of GCC at the change-log level.

Rater 2	Rater 1				
	Adaptive	Corrective	Perfective	Other	Total
Adaptive	3	1	1	0	5
Corrective	0	139	14	0	153
Perfective	0	15	128	0	143
Other	0	0	0	0	0
Total	3	155	143	0	301

As before, two different evaluators categorized the changes separately. Again for the sake of space, only one cross-rater reliability analysis is presented here. Table 10 shows the cross-rater reliability analysis for the first five versions of GCC at the change-log level. The values of all six Kappa coefficients appear in the fifth column of Table 4; all six of the Kappa coefficients were 0.80 or higher. From Table 1, we deduce that the strength of the agreement is considered excellent.

Figures 1 and 2 summarize the results of Sections 4, 5, and 6. Figure 1 shows the distribution of maintenance categories at the module level, and Figure 2 shows the distribution at the change-log level.

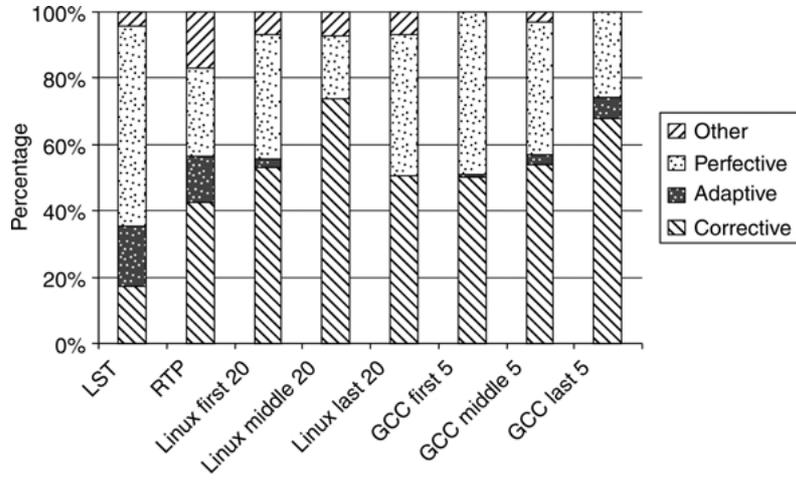


Figure 1. Distribution of maintenance categories at the module level.

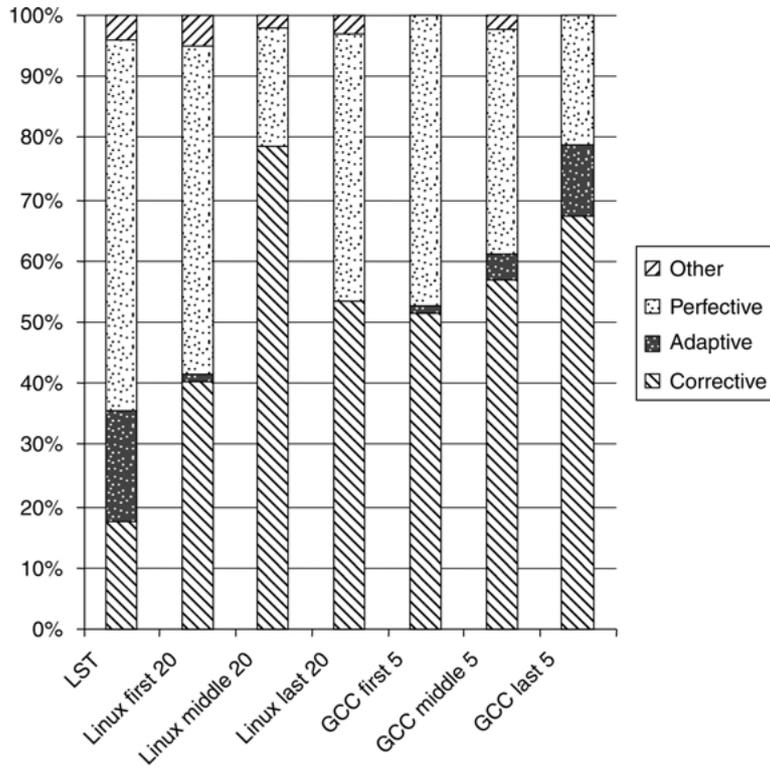


Figure 2. Distribution of maintenance categories at the change-log level.

7. Discussion

We have observed significantly more corrective maintenance than LST, and less adaptive and perfective maintenance. Table 11 shows the comparisons for the nonweighted sum of our data, at both the module level and the change-log level.

It should come as no surprise that the observed distributions of the maintenance types are different from the values predicted by LST. After all, the LST distribution was derived from a survey, whereas the distributions presented in this paper are empirical results obtained by measuring the source code itself.

What is surprising, however, is that the two distributions are so utterly different. When the same result is obtained in two different ways (from a survey and from measurements, in this instance), we do not anticipate that the two answers will be identical, but we would certainly expect that the two answers would show some similarity. Referring again to Table 11, according to the LST survey 17.4% of effort is devoted to corrective maintenance, whereas the summed data reflects 53.4% (module level) and 56.7% (change-log level), more than three times the LST value. This huge discrepancy between the measured results and the results of the LST survey needs to be understood.

One possible explanation is the LST values were obtained from data processing software, not operating systems or compilers. However, RTP is a commercial real-time product. In passing, only one of the three top-selling software engineering textbooks (Schach, 2002; Pressman, 2001; Sommerville, 2001) points out that LST cannot necessarily be extrapolated to all types of software. Sommerville (2001) restricts the result to "custom software." Furthermore, all three textbooks imply that the LST result still holds, 25 years later.

Another possible explanation is that the nature of software development has changed since 1978 as a consequence of the transition to the object-oriented paradigm. However, RTP was started in 1987, Linux in 1991, and GCC in 1985, and none of them was designed or developed as object-oriented software.

A third possible explanation for the vast discrepancy is that the LST values appertain to effort, whereas our results relate to the number of changes of each type. Graves and Mockus (Graves and Mockus, 1998) found that the effort in performing corrective maintenance is about 1.8 times greater than for comparably sized perfective maintenance. In view of the fact that all our observed percentages for corrective maintenance are already more than twice as large as the percentages predicted by LST, converting our numerical data to effort data on the basis of

Table 11. Comparison of the summed data of Tables 2, 5, 6, 8 and 9 with LST.

Maintenance category	Module level percentages (%)	Change-log level percentages (%)	LST percentages (%)
Adaptive	4.4	2.2	18.2
Corrective	53.4	56.7	17.4
Perfective	36.4	39.0	60.3
Other	0.0	2.4	4.1

Table 12. Data at the change-log level for the middle 20 versions of the Linux kernel converted to effort data.

Maintenance category	Number data		Effort data		Expected percentages (LST) (%)
	Observed number	Observed percentages (%)	Converted number	Converted percentages (%)	
Adaptive	0	0.0	0	0.0	18.2
Corrective	151	78.6	271.8	86.9	17.4
Perfective	37	19.3	37	11.8	60.3
Other	4	2.1	4	1.3	4.1
Chi-square test	$P < 0.001$		$P < 0.001$		

Graves and Mockus's conversion factor would only make the discrepancies with LST considerably worse, both at the module level and the change-log level. For example, consider the change-log level data for the middle 20 versions of Linux, shown in Table 6. (We chose this data set because there is no adaptive maintenance; no conversion factor for the effort in performing adaptive maintenance has yet been published. Also, the number of changes in category "other" is only four.) Table 12 shows the effect of scaling the corrective maintenance by 1.8 (that is, from 151 to 271.8). The deviation from LST is even more pronounced.

A fourth possible reason relates to the way that we measured maintenance activity. The LST maintenance percentages are derived from estimates of effort. In this paper, we measured maintenance activity in two ways, namely, the number of changes at the module level and the number of changes at the change-log level (where one change can span multiple modules). Although the results from the two levels of granularity are different, at both levels the observed percentages are statistically very highly significantly different from the LST percentages, and the deviations were even greater when we used Graves and Mockus' factor to convert our number data to effort data. However, it is possible that, had we measured maintenance activity in yet another way (at the code level, say), it is conceivable that the deviation from LST would not have been quite so pronounced.

A fifth possible explanation is that participants in the survey from which LST was derived simply did not have adequate data to respond to the survey. The participating software maintenance managers were asked whether their response to each question was based on reasonably accurate data, minimal data, or no data. In the case of the LST question, 49.3% stated that their answer was based on reasonably accurate data, 37.7% on minimal data, and 8.7% on no data. In fact, we seriously question whether any respondents had "reasonably accurate data" regarding the percentage of effort devoted to the categories of maintenance included in the survey, and most of them may not have had even "minimal data." In the survey, participants were asked to state what percentage of maintenance consisted of items like "emergency fixes" or "routine debugging"; from this raw information, the percentage of adaptive, corrective, and perfective maintenance was computed. Software engineering was just starting to emerge as a discipline in 1978, and it was

the exception for software maintenance managers to collect the detailed information needed. Indeed, in 1978 almost all organizations were at what we now call CMM level 1 (Schach, 2002). There is also the issue of the time needed to collect maintenance data. It took us three weeks to analyze various aspects of the changes to the 138 versions of RTP we investigated, and over nine months to analyze the 6506 Linux versions. Today, CASE environments are used in software development and maintenance, and these environments can assist in data collection. Nevertheless, our experience has been that, even when such CASE environments are used, software engineers are reluctant in the extreme to spend even a minute or two entering information that they do not view as relevant to their day-to-day tasks, even when their managers have mandated this data collection. In 1978, before such CASE tools existed, it seems most unlikely that software maintenance managers would have had much in the way of “reasonably accurate” maintenance data of any kind.

A sixth possible reason is that the managers did not tell the truth when responding to the survey. After all, corrective maintenance is performed to fix a fault; had the software been better developed, the fault would not have been present. Even though the managers were promised anonymity, it is possible that they wanted to paint their companies in a good light.

8. Conclusions and Future Work

We have examined maintenance data from three different sources, namely, RTP, a commercial real-time program, and Linux and GCC, two open-source programs. In all three cases, the distribution of maintenance categories was statistically very highly significantly different from the distribution described in Lientz et al. (1978). In particular, in every case the percentage of corrective maintenance was at least twice as large as predicted, and three times larger for the summed data.

To evaluate the reliability of our categorization results, Cohen’s Kappa coefficient was calculated for all 13 sets of categorizations. In 11 instances, the reliability was found to be excellent, and it was substantial in the other two instances.

We then reexamined the part of the survey conducted by Lientz et al. (1978) that relates to the distribution of maintenance categories. We are highly skeptical about the accuracy of the responses of the managers who participated in the survey. Furthermore, we seriously doubt that the results of a survey of software managers can ever be as accurate as empirical results based on measurements of the software itself.

We are currently examining the repeated maintenance of other software products to obtain more actual data on the distribution of maintenance categories.

Acknowledgment

This work was sponsored in part by the National Science Foundation under grant number CCR-0097056.

References

- Cohen, J. 1960. A coefficient of agreement for nominal scales. *Educ. Psych. Meas.* 20: 37–46.
- El Emam, K. 1998. *Benchmarking Kappa for Software Process Assessment Reliability Studies*. International Software Engineering Research Network Technical Report ISERN-98-02.
- GCC Home Page—GNU Project—Free Software Foundation (FSF), <http://www.gnu.org/software/gcc/gcc.html>, October 12, 2001.
- Graves, T. L., and Mockus, A. 1998. Inferring change effort from configuration management data. *Proceedings of the Fifth International Symposium on Software Metrics*, Bethesda, MD, 267–273.
- Lientz, B. P., Swanson, E. B., and Tompkins, G. E. 1978. Characteristics of application software maintenance. *Communications of the ACM* 21(6): 466–471.
- Linux Online—About the Linux operating system, <http://www.linux.org/info/index.html>, March 6, 2000.
- Mockus, A., and Votta, L. 2000. Identifying reasons for software changes using historic databases. *Proceedings of the 2000 International Conference on Software Maintenance*, San Jose, CA, 120–130.
- Nosek, J. T., and Palvia, P. 1990. Software maintenance management: Changes in the last decade. *Journal of Software Maintenance: Research and Practice* 2(3): 157–174.
- Pressman, R. S. 2001. *Software Engineering, A Practitioner's Approach*, 5th edition. Boston, MA: McGraw-Hill, 805.
- Schach, S. R. 2002. *Object-Oriented and Classical Software Engineering*, 5th edition. Boston, MA: WCB/McGraw-Hill, 10, 52, 181–189, 426.
- Schach, S. R., Jin, B., Wright, D. R., Heller, G. Z., and Offutt, A. J. 2002. Maintainability of the Linux kernel. *IEE Proceedings—Software* 149(1): 18–23.
- Sommerville, I. 2001. *Software Engineering*, 6th edition. Harlow, UK: Addison-Wesley, 606.
- Swanson, E. B. 1976. The dimension of maintenance. *Proceedings of the Second International Conference on Software Engineering*, San Francisco, CA, 492–497.
- Wang, S., Schach, S. R., and Heller, G. Z. 2001. A case study in repeated maintenance. *Journal of Software Maintenance and Evolution: Research and Practice* 13(2): 127–141.
- Weiss, N. A. 1995. *Introductory Statistics*, 4th edition. Reading, MA: Addison-Wesley.



Stephen R. Schach is an Associate Professor in the Department of Electrical Engineering and Computer Science at Vanderbilt University in Nashville, Tennessee. Steve is the author of over 100 refereed publications. He has written 10 software engineering textbooks, including *Object-Oriented and Classical Software Engineering*, Sixth Edition, to be published by WCB/McGraw-Hill in 2004. He consults internationally on software engineering topics. Steve's current research interests are empirical software engineering, software maintenance, and open-source software engineering. He obtained his PhD from the University of Cape Town in South Africa.

Bo Jin obtained his MS in Computer Science at Vanderbilt University in Nashville in Tennessee in 2002. His main research interest is in software maintenance.



Ligu Yu is a PhD student of Vanderbilt University, EECS department. His research topic concentrates on the maintainability of the Linux kernel and open source-software development. Before working on software engineering, his research focused on modeling and system identification, fault detection and isolation of hybrid systems.



Gillian Z. Heller is a Senior Lecturer in the Department of Statistics at Macquarie University, Sydney, Australia, where she has been for the last 10 years. Her BSc (Hons) and PhD degrees are in Mathematical Statistics, from the University of Cape Town, South Africa, and her MSc in Operations Research is from the University of South Africa. Gillian's research interests are in discrete distribution theory, with applications in biostatistics.



Jeff Offutt is an Associate Professor of Information and Software Engineering at George Mason University and holds part-time visiting positions at NIST and Skövde University. His current research interests include software testing, analysis and testing of web applications, software maintenance and object-oriented program analysis. He has published over 75 refereed papers. He was program chair for ICECCS 2001 and is on the editorial boards for the *IEEE Transactions on Software Engineering*, the *Journal of Software Testing, Verification and Reliability*, the *Journal of Software and Systems Modeling* and the *Software Quality Journal*. His PhD is from the Georgia Institute of Technology. He previously held a faculty position in the Department of Computer Science at Clemson University.