William B. Frakes and Christopher J. Fox

Sixteen Questions About **Software Reuse**

oftware reuse is the use of existing software knowledge or artifacts to build new software artifacts. Reuse is sometimes confused with porting. The two are distinguished as follows: Reuse is using an asset in different systems; porting is moving a system across environments or platforms. For example, in Figure 1 a component in System A is shown used again in System B; this is an example of reuse. System A, developed for Environment 1, is shown moved into Environment 2; this is an example of porting.

Many organizations are implementing systematic reuse programs and need answers to practical questions about reuse. Little empirical data has previously been collected to help answer these questions. In this article we answer 16 questions that, in our experience, are commonly asked about reuse, using survey data we collected recently from organizations in the U.S. and Europe [5]. Our analysis supports some commonly held beliefs about reuse and contradicts some others.

Survey Respondents

We surveyed software engineers, managers, educators, and others in the software development and research community about their attitudes, beliefs, and practices in reusing code and other lifecycle objects. Survey respondents were drawn arbitrarily from the software engineering community. A total of 113 people from 28 U.S. organizations and one European organization responded to the survey during '91-'92.

The survey respondents do not form a large random sample of the software engineering community. Nevertheless, indicators of the experience, education, and background of the respondents suggest that they are fairly representative of experienced software engineers and managers at high-technology companies in the U.S. Seventy-seven percent of survey respondents are software engineers or managers. Most respondents have considerable software engineering experience, with a mean of 12.2 years of work experience on 9.2 projects in 3.3 organizations. Fifty percent of respondents have degrees in computer and information science. Most have bachelor's or master's degrees in computer science, electrical engineering, or mathematics.

Of the 29 organizations represented in the survey, 18 are represented by a single respondent. Five organizations have two respondents, and six have three or more respondents. The maximum number of respondents from a single organization is 26. Six organizations in this survey are universities. The rest are high-technology research companies in scientific and technical fields with significant software development efforts. Respondents work in organizations spanning the range of company, division, and project sizes, with a median company size of 25,000. These individuals work for companies with as few as 35 and as many as 350,000 employees. This helps ensure that our conclusions are generalizable to companies of various sizes.

Most respondents work for companies at the cutting edge of software technology, particularly in software and aerospace companies, which account for 59% of the respondents. The most frequently used language in respondents' companies is C, followed by Fortran, Pascal, and Ada.

The Questions

The 16 questions we chose are some we have found to be commonly asked by organizations attempting to implement systematic reuse. The answers to many of these questions are often taken for granted in the software engineering community, but have not been verified empirically. Our answers to these questions are based on analysis of survey data and hence are empirically based.

QUESTION 1: *How widely reused are common assets?*

Software engineers have many reusable assets available to them, but do they actually use them and find them valuable? Our data says that the answer is yes and no. Some assets, such as those in the Unix environment, are widely used and perceived to be valuable. Other assets, such as the Cosmic collection from NASA, are not used by many software engineers and are not perceived to be very valuable by their users. Most assets were perceived, on average, to be at least somewhat valuable by respondents who had used them, suggesting that these asset collections are generally helpful to their user communities.

Table 1 shows the number of respondents using each asset. This is our measure of the amount of use; table entries are ranked by this measure, with the most widely used assets listed first. Respondents rated the assets using a five-point scale, from 1 (Not Valuable) through 3 (Somewhat Valuable) to 5 (Very Valuable). The median rating for each asset is our measure of perceived value.

There are many possible reasons for the variability in the use and perceived value of these assets, including lack of information and education, asset quality, and ease of access to the assets. The reasons why some asset collections are more popular and perceived as more valuable are not known, but anecdotal evidence in the reuse community suggests that relevant functionality is an important factor.

QUESTION 2: *Does programming language affect reuse?*

Many organizations believe that they need to change their programming language to promote reuse. However, opinion on the importance of the choice of programming language for reuse is divided. Some people think that language is of little or no importance for reuse, while the adherents of languages such as Ada, C++, Smalltalk, and Eiffel argue that features of these languages (e.g., support for abstraction, inheritance, strong typing) provide better reuse support. Past analyses of the effects of language on various aspects of the development process have shown that programming language may not be very important. For example, Boehm's studies show that the rate of

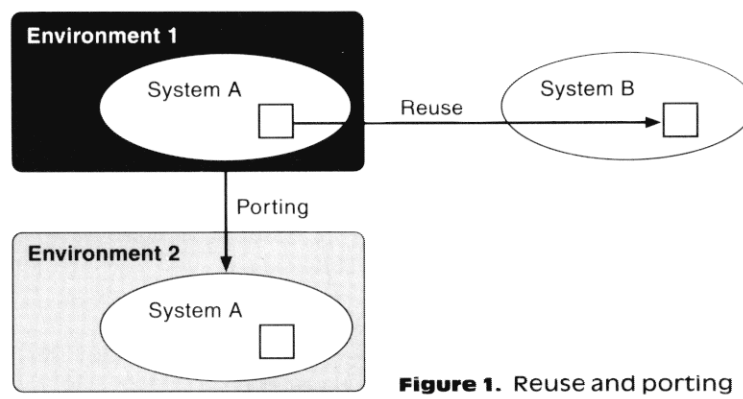


Figure 1. Reuse and porting

Table 1. Amount of use and perceived value of common assets

Assets	Users	Never used	Percent who used	Perceived value
UNIX tools	70	31	69	4
Program templates	65	32	67	3
Document templates	63	32	66	4
FORTRAN libraries	53	45	54	4
X widgets	45	53	46	4
Ada math library	27	69	28	4
Booch	23	73	24	3
4 GL	20	74	21	3.5
Grace	9	86	10	3
Cosmic	7	85	8	2

production of source code statements is relatively independent of language and even language level [1].

We used our data to answer the question of whether programming language affects reuse by correlating the usage of 11 common programming languages with levels of organizational code reuse. Language usage rankings were obtained by having respondents rank order the languages used in their companies, from 1 for the most commonly used language up to 11 for the least commonly used language. To obtain reuse levels, respondents were asked, "What percent of the lifecycle objects your organization creates are typically composed of reusable parts?" The responses were used as the measure of organizational code reuse level. Thus, each subject provided a usage ranking value (1 to 11) for each language and an associated level of

Table 2. Choice of language and code reuse levels

Language	Correlation: Language usage and organizational code reuse
Jovial	0.00
Smalltalk	0.00
Pascal	0.03
Cobol	0.09
C	0.09
Ada	0.10
Fortran	0.17
C++	0.20
Assembler	0.25
Lisp	0.28
PL-1	0.69

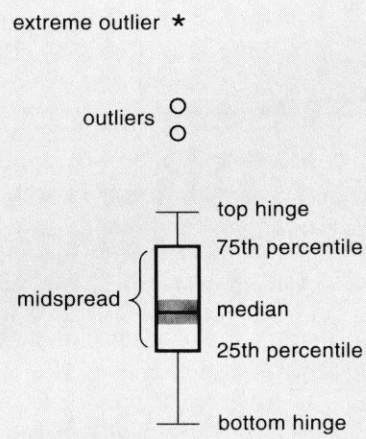
Statistical Methods Used

Much of the analysis reported here is of correlations between variables. We used the Spearman correlation, a nonparametric measure that treats data as ranks [13]. Nonparametric statistics are often used because they require fewer assumptions about the data. There are several considerations in interpreting correlations. First is the effect size, or magnitude, of the correlation. In research outside the physical sciences, correlations of 0.5 and above are often considered large, those of 0.3 to 0.5 are considered medium, and those of 0.1 to 0.3 are considered small [2]. We follow this classification in this article.

A related issue is statistical significance, which addresses whether an observed statistic might have occurred by chance alone. Statistical significance is a function of sample size, effect size, and the statistical measure used. We use a significance level of 0.05 in this study, which means that we regard a finding as real only if the probability that it is a statistical fluke is less than 0.05.

We also summarize data and compare groups using *boxplots* [12], or graphical representations of the distribution of a set of data (See figure). The bottom of the box is the 25th percentile and the top of the box the 75th percentile. Thus, half of the data points in the set fall within the box. The difference between the 75th and 25th percentiles is the *midspread*. The *top hinge* is the largest data point 1.5 midspreads or less above the 75th percentile; the *bottom hinge* is the smallest point 1.5 midspreads or less below the 25th percentile. An *outlier* is any data point above or below a hinge, but no more than 3 midspreads above or below. An *extreme outlier* lies more than 3 midspreads from the hinge.

The line across the box is the median, or 50th percentile.



The shaded area around the median is the 95% confidence interval for the median. It is placed symmetrically around the median according to the formula:

$$median \pm (1.57 \times midspread \div \sqrt{n})$$

where n is the sample size.

Boxplots reveal much about the shapes of data distributions. The length of the box and the placement of hinges and outliers shows the dispersion of the data, and the median line shows its center. The placement of the median line in the box and the distance of the hinges from the box show how skewed the data is. Side-by-side boxplots illustrate the relationships of these characteristics for two or more distributions. The confidence bands around the medians provide a test for the difference of medians: if the bands don't overlap, then the medians are significantly different [9].

code reuse (0 to 100).

Table 2 shows the Spearman correlations between the usage of the 11 programming languages and the levels of code reuse. Because languages are ranked on a scale in which lower numbers indicate more usage, a high negative correlation between usage ranking and code reuse would indicate that a language promotes reuse.

Only one of these correlations—for PL/I—is statistically significant at the 0.05 level. This strong positive correlation for PL/I suggests that using PL/I may retard code reuse. On the other hand, usage of languages usually thought to promote reuse, like Ada and C++, shows no significant correlation with code reuse levels. We also found that higher-level languages are no more strongly correlated with high reuse levels than is assembly language. Our conclusion is that choice of programming language does not affect code reuse levels. The implication of this result is that, contrary to popular belief, efforts to increase reuse levels should focus on other factors besides programming language.

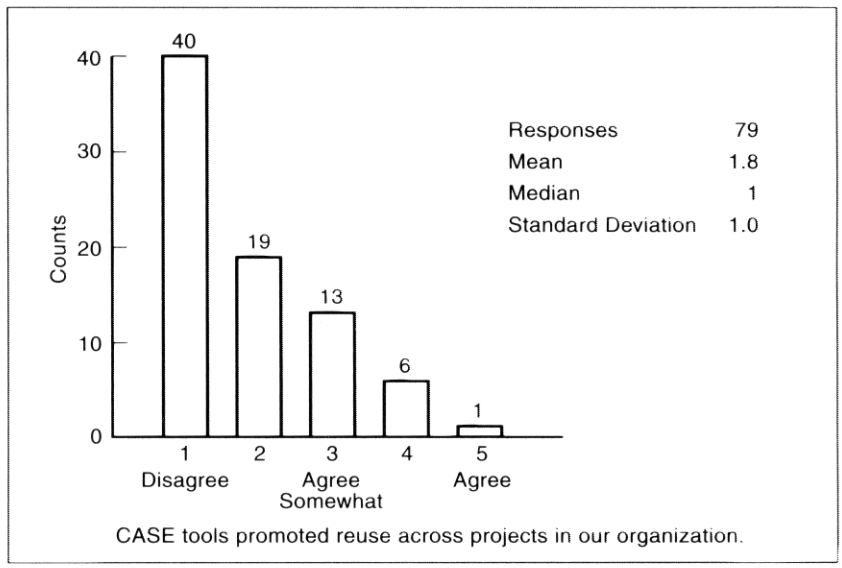


Figure 2. CASE tools and promoting reuse

QUESTION 3: *Do CASE tools promote reuse?*

The large literature on reuse CASE tools and their growing market show that many organizations regard CASE tools as a way to improve reuse. To study this question, respondents were asked whether they agreed with the statement, "CASE tools have promoted reuse across projects in our organization." The responses are shown in Figure 2.

The data shows that respondents generally feel that CASE tools have not promoted reuse across projects in their organizations; 75% of respondents do not agree even somewhat that CASE tools have promoted reuse.

We further explored this issue by running Spearman correlations between respondents' degree of belief that CASE tools have promoted reuse and their responses to the question, "What percent of the lifecycle objects your organization creates are typically composed of reusable parts?" The lifecycle objects considered were requirements, designs, code, test plans, test cases, and user documentation. We found no significant correlations at the 0.05 level. Because reuse levels of lifecycle objects are not significantly higher in organizations where CASE tools are thought to promote reuse than in organizations where they are not thought to promote reuse, there is no evidence that CASE tools promote reuse.

We conclude that CASE tools are not currently effective in promoting reuse. There are at least three reasons why this may be so: Reuse CASE tools may not be used; they may not be used correctly; or they may not be effective in promoting reuse even when they are used correctly. This area needs further investigation.

QUESTION 4: *Do developers prefer to build from scratch or to reuse?*

Many people believe that software engineers prefer to build their own software rather than reuse someone else's. This is often referred to as the "Not Invented Here" (NIH) syndrome.

We investigated this question by looking at respondents' answers to the statement, "It's more fun to write my own software than to try to reuse." Responses are shown in Figure 3.

Most respondents (72%) do not have the NIH syndrome. We conclude that most developers prefer to reuse rather than build from scratch. This result contradicts conventional wisdom in the software engineering community, but is in agreement with the findings of another recent study [3].

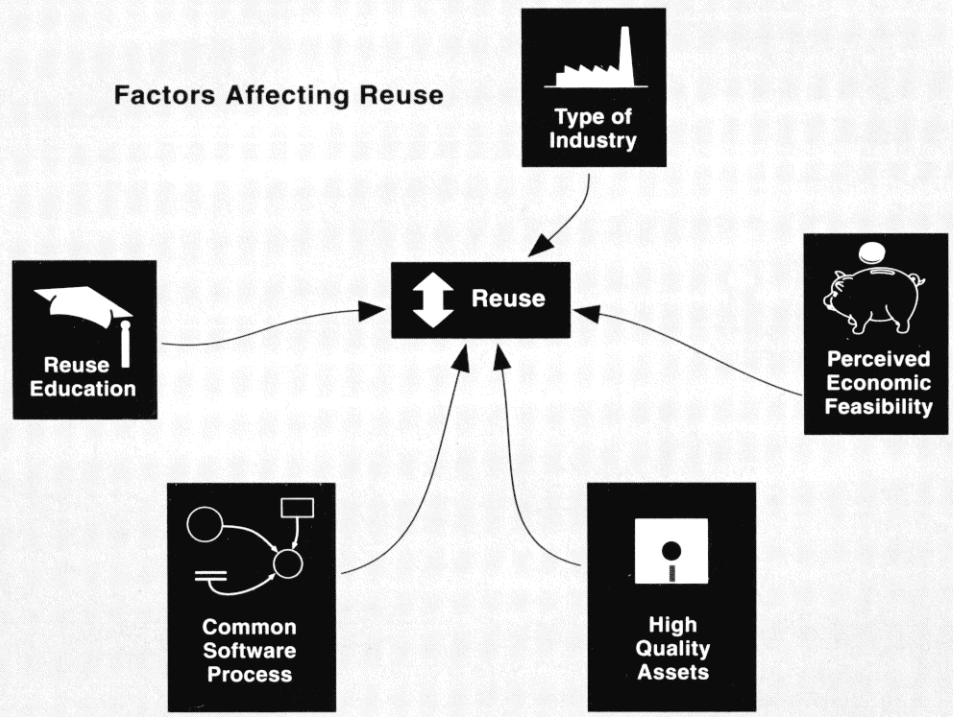
QUESTION 5: *Does perceived economic feasibility influence reuse?*

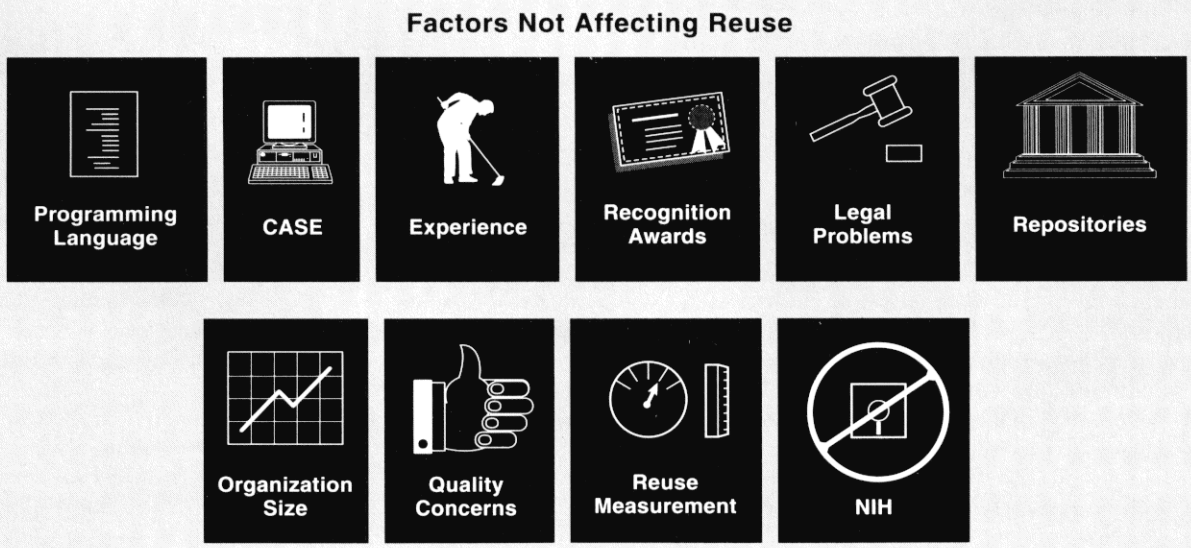
Reuse may not be done if it is not believed to be economically feasible. We examined whether perceived economic feasibility influences reuse by comparing levels of individual and organizational code reuse with respondents' agreement with the statement, "Reuse is economically feasible in my organization." We first analyzed the distributions of individual and organizational code reuse levels for each level of agreement that reuse is economically feasible. Figure 4 summarizes these distributions with boxplots. The boxplots show a clear trend toward higher reuse as belief in the economic feasibility of reuse increases.

This trend was verified by correlating code reuse levels for individuals and organizations with degree of be-

How to Improve Reuse

This figure summarizes the effects on systematic reuse of the factors we investigated. An organization trying to improve systematic reuse should concentrate on education about reuse, developers' understanding of the economic feasibility of reuse, instituting a common development process, and making high-quality assets available to developers. The other factors, despite the conventional wisdom, do not seem to be important. It should be understood, however, that these conclusions are based on data gathered from across the industry—the factors salient for a particular organization may be different. The best course is to investigate the factors affecting reuse in the target organization (through surveys, case studies, or other techniques) and take action based on the results.





lief in the economic viability of reuse. For both individuals and organizations we found that higher perceptions of economic viability correlated significantly at the 0.05 level with code reuse levels. The correlation between individual code reuse and perceived economic viability was 0.35; the correlation between organizational code reuse and perceived economic viability was 0.39. Both of these are medium-strength correlations, so we conclude that perceived economic feasibility does influence reuse.

This result implies that it is important to convince software engineers that reuse is economically justified. Management must bear the responsibility for educating software development staff that reuse is a desirable and economically viable practice.

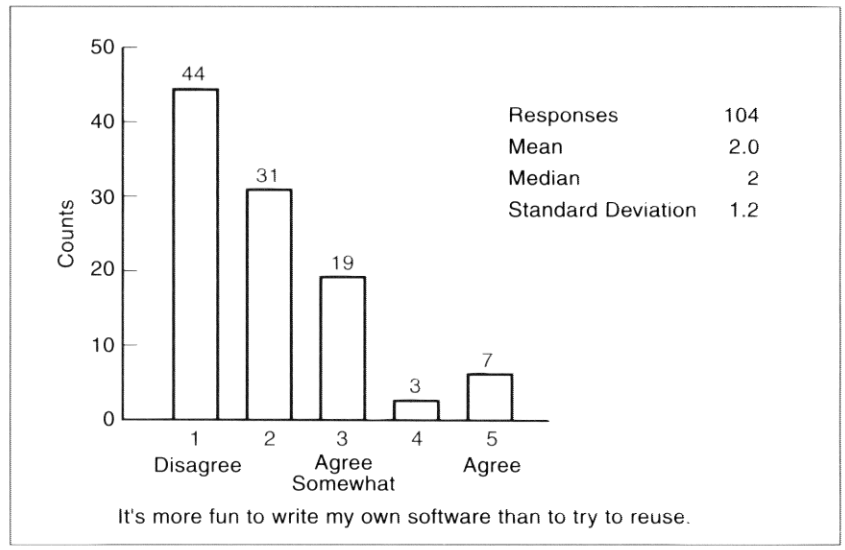
QUESTION 6: *Does reuse education influence reuse?*

We considered the influence of education on reuse from two perspectives. First we looked at respondents' answers to the statement, "I was educated about software reuse in school" compared to individual levels of lifecycle object reuse. We found that education does influence code and design reuse levels, as shown in Figure 5. People who were educated about reuse in school reported significantly higher median levels of code and design reuse at the 0.05 level.

Despite the importance of reuse education in school to reuse success, we found that relatively few of our respondents—only 13 (17%) of the 76 who responded to this question—had been educated about reuse in school.

Because most software engineers working today were not educated about reuse in school, it is up to industry to train them. We next examined respondents' agreement with the statement, "My organization has an education program about software reuse," and compared this with organizational reuse levels of lifecycle objects. We found that organizations with a corporate reuse education program had significantly higher median levels of code reuse at the 0.05 level, as shown in Figure 6. This finding supports the importance of corporate reuse education. Unfortunately, corporate reuse education is also rare, with only 15 (19%) of 78 respondents reporting that their organization has a reuse training program.

We conclude that education about reuse, both in school and at work, improves reuse and is a necessary part of a reuse program, though reuse education is still relatively rare in both academia and industry. Management must again bear the responsibility of ensuring that software development staff is trained in reuse if systematic reuse programs are to succeed.



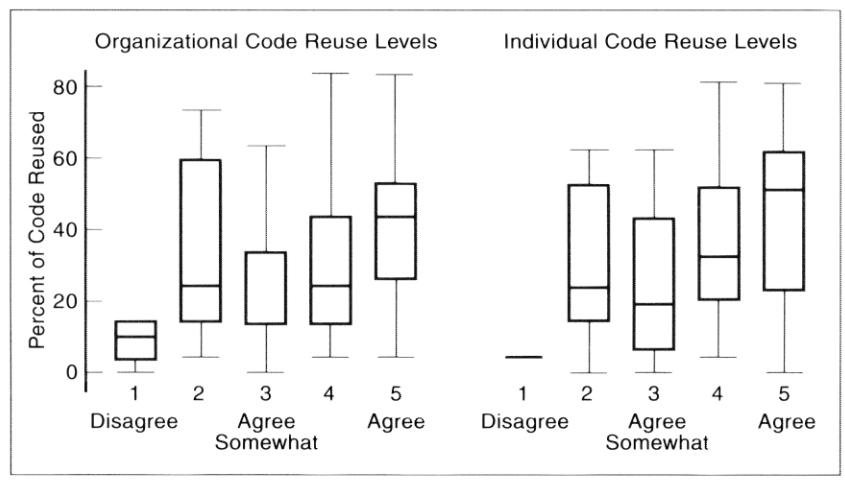
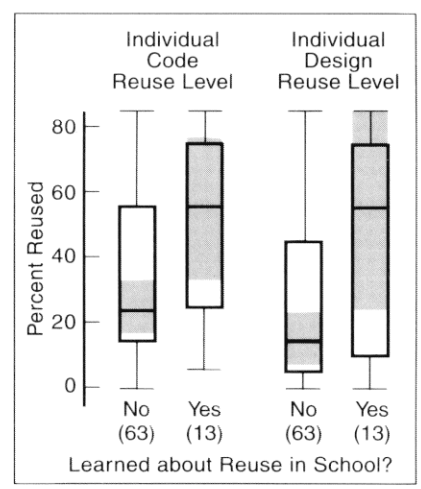
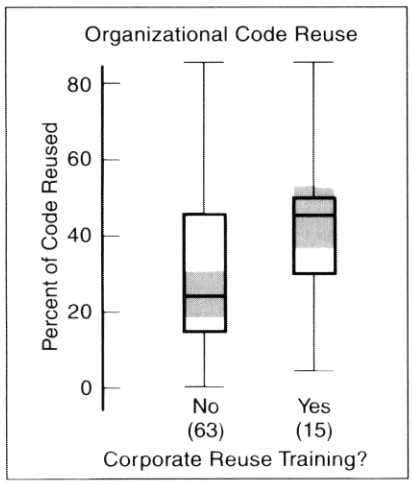


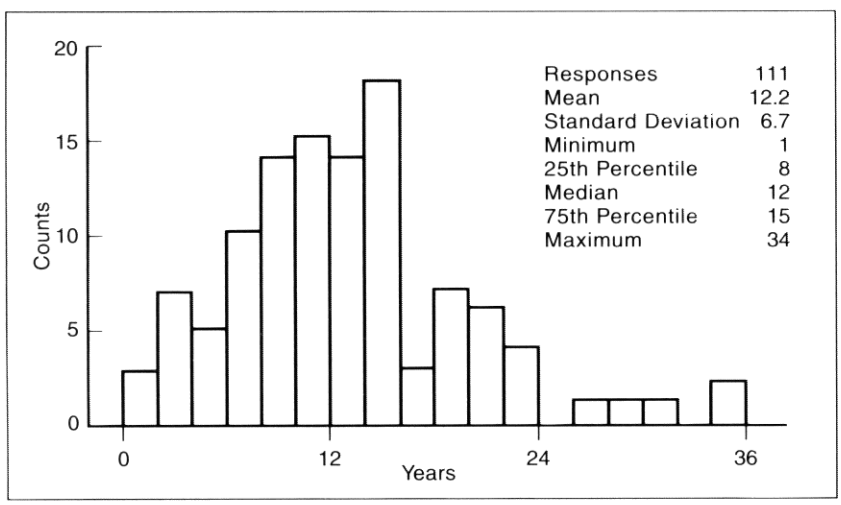
Figure 3. Software creation versus reuse preferences

Figure 4. Perceived economic feasibility versus code reuse levels

Figure 5. Learning about reuse in school and reuse level







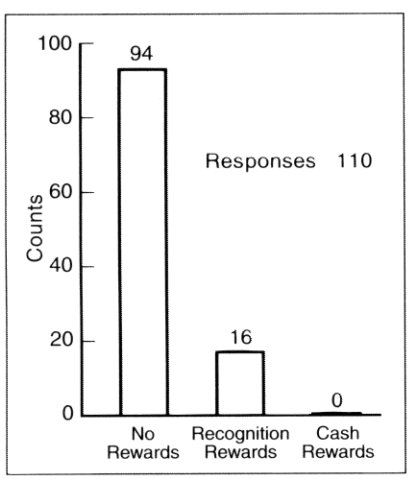


Figure 6. Corporate reuse training and reuse level

Figure 7. Respondents' years of experience

Figure 8. Rewards for reuse

QUESTION 7: *Does software engineering experience influence reuse?*

It is often thought that more experienced software engineers are better practitioners. To explore this question with regard to reuse practice, respondents were asked to report their years of experience in software engineering. The distribution of the results is shown in Figure 7.

Respondents tend to be quite experienced in the field, with a mean of 12.2 years of experience. Half the respondents have between eight and 15 years' experience. We ran Spearman correlations between years of software engineering experience and personal reuse levels for lifecycle objects. We found no correlation between them at the 0.05 level.

We conclude from this that software engineering experience has no effect on reuse of lifecyle objects. This somewhat surprising result may be attributable to the historical lack of training in reuse (see discussion of Question 6) and to the fact that systematic reuse has only recently become a salient goal of many software organizations.

QUESTION 8: *Do recognition rewards increase reuse?*

Reuse incentives have been reported to be necessary catalysts for reuse. Two kinds of rewards have been tried: recognition rewards and cash rewards. GTE, for example, reported using cash payments to producers of reusable assets in building its successful reuse effort [10]. Nippon Novel pays software engineers several cents per line of code registered in a reuse repository and several cents per reused module [6]. Our respondents said that rewards for reuse are rare. No respondent reported cash bonuses, and only a few (15%) report any kind of recognition, as shown in Figure 8.

To investigate the question of whether recognition rewards increase reuse, we compared boxplots of the levels of organizational reuse of lifecycle objects for groups of respondents reporting no rewards and recognition rewards. We found no significant differences between them. The results (for code) in Figure 9 are typical of the analysis, showing no significant difference at the 0.05 level between the no-reward and recognition groups.

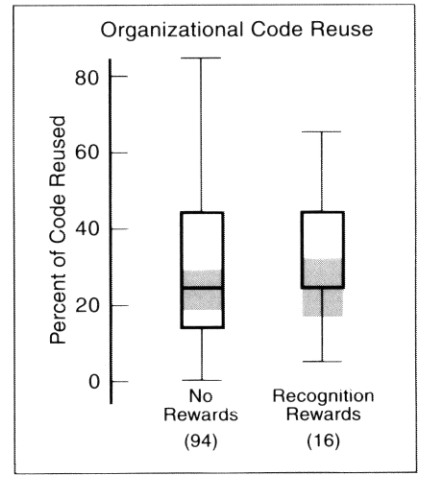


Figure 9. Code reuse levels and recognition rewards

We ran the same analysis for individual reuse levels and also found that recognition rewards made no significant difference. Our results contradict the common belief that recognition is a sufficient reward for reuse. It may be that only monetary rewards are sufficient motivators.

This is in line with the GTE experience that money is a needed reuse motivator. Unfortunately, the lack of respondents in organizations with cash rewards made it impossible to investigate this question in our study.

QUESTION 9: *Does a common software process promote reuse?*

Study and improvement of the software process has been a popular topic in the software engineering community in recent years as a consequence of work at the Software Engineering Institute [8], prompting the question whether the software process affects software reuse. We asked respondents whether they agreed with the statement, "A common software development process has promoted reuse across projects in our organization." The responses are shown in Figure 10.

The data shows that respondents generally do not agree that common software processes have promoted reuse across projects in their organization. This might mean either that their organizations do not have a defined process or that the process has failed to support reuse. Recent studies of software process capability show that most organizations have immature processes (level 1 on the Software Engineering Institute [SEI] process maturity scale), supporting the view that organizations typically lack a defined process.

To investigate question 9 further, we ran Spearman correlations between the degree to which respondents agreed that a common process promoted reuse and their organizational reuse levels. These correlations were significant at the 0.05 level and are shown in Table 3.

The correlations shown in Table 3 range in strength from weak (for requirements, code, and user documentation) to moderate (for designs, test plans, and test cases). These correlations demonstrate that there is more reuse in organizations with a common software process that promotes reuse. We thus conclude that a defined software process that promotes reuse does affect software reuse levels. The consequence of this conclusion is that gains in process maturity can translate into gains in software reuse.

Figure 10. A common software process and reuse levels

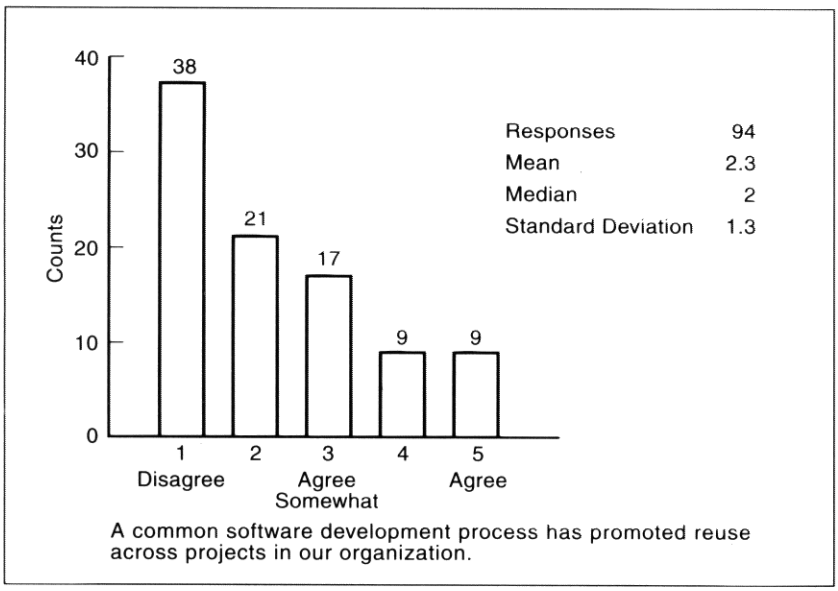


Table 3. Organizational reuse levels and a common software process

Organizational reuse level	Correlation between reuse levels and agreement that a common software process promotes reuse across projects
Requirements	0.24
Design	0.40
Code	0.24
Test plans	0.31
Test cases	0.34
User documentation	0.15

QUESTION 10: *Do legal problems inhibit reuse?*

Legal problems are thought to be a serious impediment to reuse. Legal issues regarding contracting, ownership, and liability for reusable components are still unresolved. To test whether these legal problems affect reuse, respondents were asked whether they are inhibited by possible legal problems. Responses are shown in Figure 11.

Legal problems do not appear to be an impediment for most survey respondents, 68% of whom agree less than somewhat that they are inhibited by legal problems.

Spearman correlations were run between levels of reuse of lifecycle objects for individuals and organizations and reported inhibition by legal problems. We found no significant correlations other than a weak negative correlation with levels of reuse of user documentation. Hence it appears that people are not inhibited in their reuse practices by fears of legal problems. Today most reuse goes on within companies, so legal issues are of less concern. This may change as

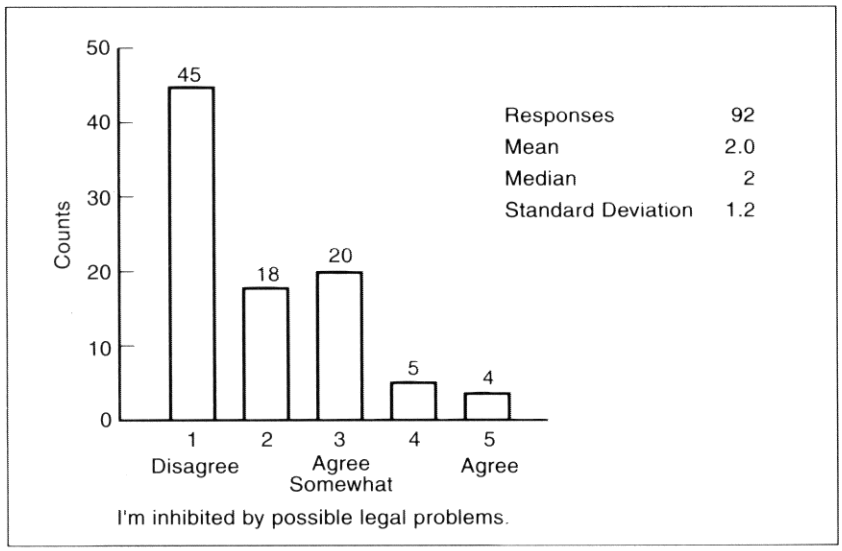


Figure 11. Reuse and legal problems

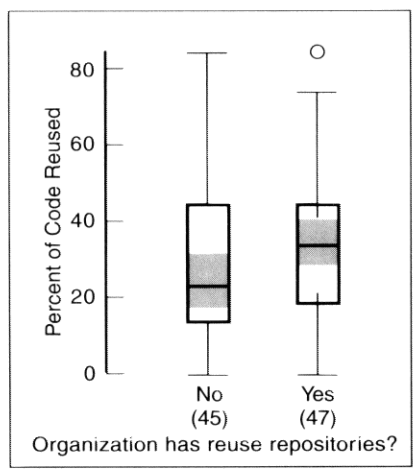
reusable assets are increasingly marketed outside companies.

QUESTION 11: *Does having a reuse repository improve code reuse?*

A reuse repository is a collection of reusable assets, along with a searching mechanism for locating assets meeting development needs. The importance of a repository for promoting reuse has been the subject of debate. Many organizations have considered a repository central to their reuse efforts, and many kinds of repository mechanisms have been reported [7]. On the other hand, Tracz has argued that repositories are not of critical importance for reuse [11].

We examined the impact of having a repository on organizational code reuse levels, with the results shown in Figure 12. We found organizations with a repository have median code reuse levels 10 percent higher than organizations that do not have reuse repositories, but this difference is not statistically significant at the 0.05 level. Our analysis does not take into account what type of repository or what type of indexing was used.

We conclude that having reuse repositories does not improve levels of code reuse. Organizations trying to improve systematic reuse should probably not focus on repositories in their improvement efforts, at least initially.



QUESTION 12: *Is reuse more common in certain industries?*

Respondents were asked to classify their companies by primary business. The results are shown in Figure 13.

Most of the respondents work for companies in high-technology industries such as software (34%), aerospace (25%), manufacturing (14%), and telecommunications (6%). Some of the software systems these companies build push the limits of available technology. The "Other" category (21%) includes university respondents and respondents from companies in electronic instrumentation and equipment manufacturing and in telemetry collection and management.

The boxplots in Figure 14 show that there are significant differences in the reuse of lifecycle objects between different industries. In particular, the telecommunications industry has significantly higher levels of reuse in several cases, and the aerospace industry tends to have significantly lower levels in several cases.

We conclude that there are significant differences in reuse levels of various lifecycle objects in different industries, with telecommunications leading in reuse and aerospace trailing. The reasons for these results are

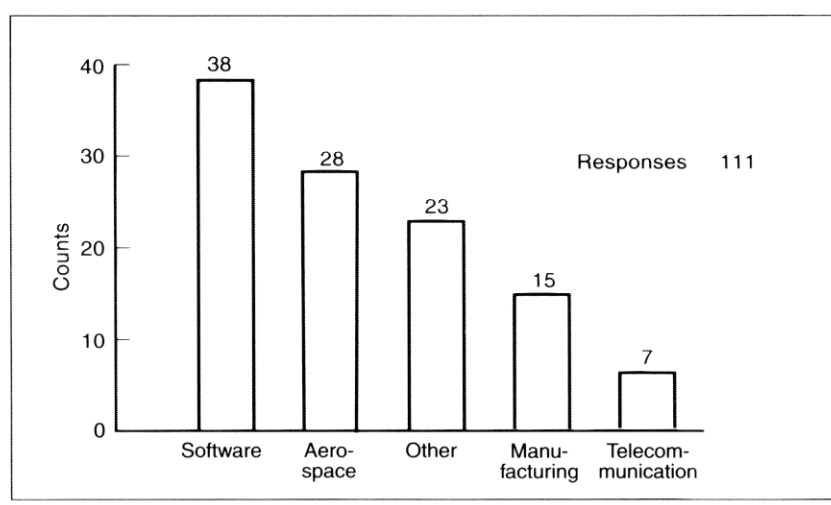


Figure 12. Reuse repositories and code reuse levels

Figure 13. Distribution of respondents across industries

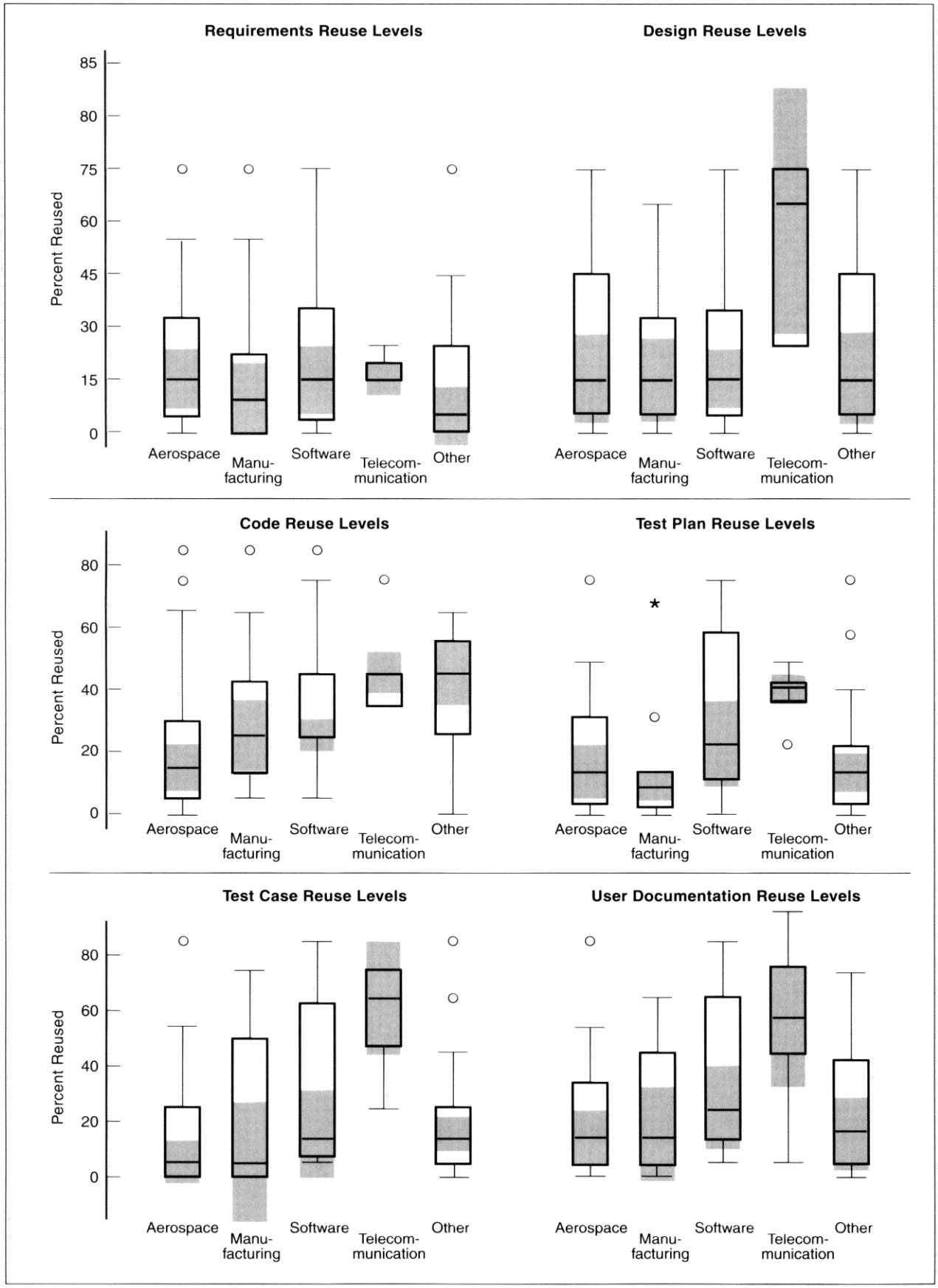


Figure 14. Reuse levels in various industries

not currently known, but they suggest a course of action for lagging industries: Industries with low reuse levels might benefit from studying and adopting reuse practices in industries that lead in software reuse.

QUESTION 13: *Are company, division, or project sizes predictive of organizational reuse?*

Small organizations sometimes wonder whether systematic reuse is a realistic goal, given the scope of their domains and limits on their resources. On the other hand, large organizations sometimes feel that instituting systematic reuse is unrealistic because of the large investments of resources and time required. We considered this issue by examining the question of whether organization size is predictive of amount of reuse.

Survey respondents come from companies that run the gamut from those with a few employees to one with over 350,000 employees. Similar variations are observed for division and project sizes. Because the distributions are very skewed (not symmetric around the mean), the median is a better summary of typical values. The median number of employees in respondents' companies is 25,000. The median division size is 350, and the median project size is seven.

We tested the hypothesis that company, division, and project sizes influence organizational reuse levels by running Spearman correlations between reported organization sizes and reuse levels for all lifecycle objects. No significant correlations were found. We concluded that company, division, and project sizes are not predictive of reuse levels. This suggests that organizations of any size may succeed (or fail) to institute systematic reuse.

QUESTION 14: *Are quality concerns inhibiting reuse?*

Reuse is likely to occur only if potential reusers are confident of the quality of reusable assets. It is commonly thought that software engineers distrust assets developed outside their immediate environment, and thus are less likely to reuse assets from outside their organization. This is another aspect of the NIH syndrome. We considered several responses in investigating this issue. Respondents were asked to rate their agreement with the statement, "Software developed elsewhere meets our standards," and were also asked, "What percentage of the parts you reuse are from external sources?" A Spearman correlation between these variables showed no relationship, suggesting that quality concerns were not related to amount of external reuse.

Respondents were also asked to rate their agreement with the statement, "I've had good experience with the quality of reusable software." Responses to this question are summarized in Figure 15, which shows that experiences have generally been favorable, with 67 respondents (69%) agreeing at least somewhat that their experiences have been good.

Spearman correlations between the responses shown in Figure 15 and the respondents' reported personal levels of reuse showed no relationship, except for a weak correlation with reuse of requirements.

These results suggest that satisfaction with the quality of reusable assets has no influence on reuse levels. This result does *not* mean that asset quality is unimportant, but that the assets encountered by respondents have generally been of sufficient quality to meet their needs. This situation may not persist if asset quality declines or if user expectations increase.

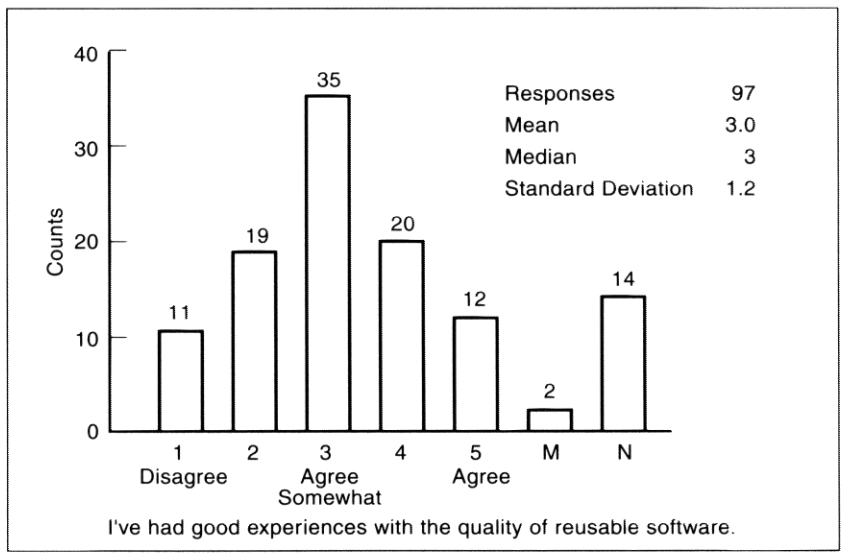
QUESTION 15: *Are organizations measuring reuse, quality, and productivity?*

Reuse measurement is crucial for determining if a reuse program is succeeding [4], but we found that few organizations currently are measuring reuse. We asked respondents if their organization has a program in place to measure level of reuse. The respondents' answers are shown in Figure 16. Only 16 respondents (14%) said their organization is currently measuring reuse, and 12 respondents (11%) did not know if their organization is measuring reuse. Thus, at most 25% of respondents are in organizations that measure reuse.

Respondents were also asked whether their organization has a program in place to measure software quality. Responses to this question are shown in Figure 17.

Quality is more widely measured than reuse: Fully 47 respondents (42%) said that their organizations have programs in place to measure software quality.

Figure 15. Experiences with the quality of reusable software



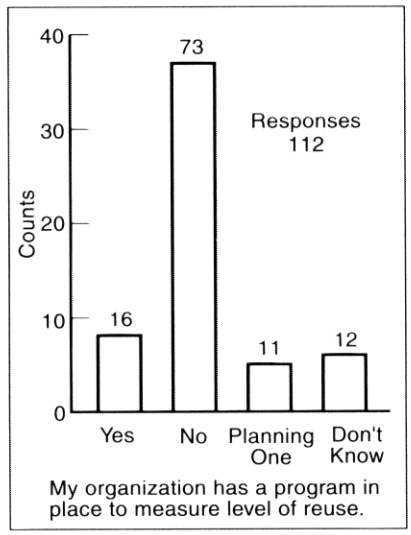
Finally, respondents were asked whether their organizations have programs in place to measure software productivity. Figure 18 shows the results of this question.

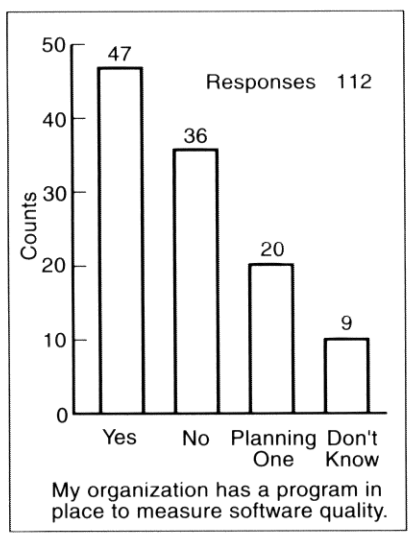
Respondents report that productivity measurement is relatively rare. Only 32% of respondents report productivity measurement programs in place, and only 12% report that one is planned.

We conclude that measurement of reuse levels, software quality, and software productivity are not done in most organizations, and thus that these organizations cannot be properly managing their software processes and products, including reuse.

QUESTION 16: ***Does reuse measurement influence reuse?***

Many people have claimed that measuring an activity will tend to increase it. This is reflected in the common saying, "If you aren't measuring it, it's getting worse." We examined this question by looking at the differences between median levels of organizational reuse for various lifecycle objects for respondents whose organizations are and are not measuring reuse. While reuse measurement is needed to manage a systematic reuse program, according to our data it is not correlated with reuse levels of any organizationally created lifecycle objects. In other words, it appears that organizations that measure reuse are not using these measurements to improve reuse levels. We found no significant differences at the 0.05 level in median levels of reuse between organizations that do and those that do not measure reuse.





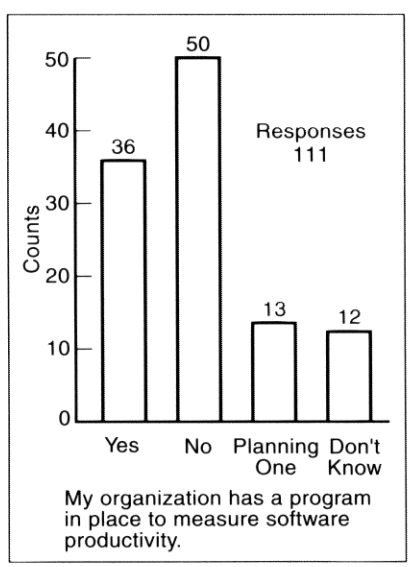


Figure 16. Respondent organizations measuring reuse

Figure 17. Respondent organizations measuring quality

Figure 18. Respondent organizations measuring productivity

Summary

Table 4 summarizes our answers to the 16 questions about reuse based on the reuse survey. We found that while some common reusable assets, such as the Unix tools, are widely used and highly regarded by software engineers, others, such as the Cosmic collection, are not. We also found that most software engineers would prefer to reuse software rather than build it from scratch, contradicting the common wisdom that software engineers prefer building things themselves rather than reusing.

Though adherents of a given programming language often claim that one language supports reuse better than others do, we found no evidence for this, nor for claims that CASE tools or software repositories promote reuse. On the other hand, our data shows that reuse education, both academic and industrial, is important for improving reuse, as is the perception that reuse is economically viable. The belief that a common software process promotes reuse also improves reuse.

We did not find that software reuse increases with more software engineering experience, nor that legal problems are a serious reuse impediment. We found that reuse levels were significantly higher for some lifecyle objects in telecommunications than in other fields, especially aerospace. We found no relationship between organization size and level of reuse. Our respondents were not influenced in their reuse efforts by concerns about asset quality. Few organizations measure reuse, quality, and productivity, but measurement of reuse is not correlated with reuse levels, though of course reuse measurement is needed to manage a systematic reuse program.

Acknowledgments

We wish to thank Steven Wartik and Gloria Hasslacher for careful reviews and many suggestions for improvement. We would also like to thank the *Communications* reviewers who made comments that helped us improve this article. ◘

References

1. Boehm, B. *Software Engineering Eco-*

Table 4. Answers to the sixteen questions

Questions	Answers	Notes
1. How widely reused are common assets?	Varies	Some (e.g., the Unix tools) are widely reused others (e.g., Cosmic) are not.
2. Does programming language affect reuse?	No	
3. Do CASE tools promote reuse?	No	
4. Do developers prefer to build from scratch or to reuse?	Prefer to reuse	
5. Does perceived economic feasibility influence reuse?	Yes	
6. Does reuse education influence reuse?	Yes	Corporate training is especially helpful.
7. Does software engineering experience influence reuse?	No	
8. Do recognition rewards increase reuse?	No	
9. Does a common software process promote reuse?	Probably	Respondents say no, but reuse levels suggest belief in the efficacy of a common process helps.
10. Do legal problems inhibit reuse?	No	May change in the future.
11. Does having a reuse repository improve code reuse?	No	
12. Is reuse more common in certain industries?	Yes	More common in telecommunications, less in aerospace.
13. Are company, division, or project sizes predictive of organizational reuse?	No	
14. Are quality concerns inhibiting reuse?	No	May change in the future.
15. Are organizations measuring reuse, quality, and productivity?	Mostly no	
16. Does reuse measurement influence reuse?	No	Measurements probably not being used.

nomics. Prentice-Hall, Englewood Cliffs, N.J., 1981.

2. Cohen, J., and Cohen, P. *Applied Multiple Regression/Correlation Analysis for the Behavioral Sciences*. Erlbaum, Hillsdale, N.J., 1975.
3. Favaro, J. What price reusability? A case study. *Ada Letts. 11*, 3 (1991), 115–124.
4. Frakes, W.B. Software reuse as industrial experiment. *Am. Program. 6*, 9 (1993), 27–33.
5. Frakes, W.B., and Fox, C.J. *Software Reuse Survey Report*. Software Engineering Guild, Sterling, Va., 1993.
6. Frakes, W.B., and Isoda, S. Success factors of systematic reuse. *IEEE Softw. 11*, 5, (May 1994), 15–19.
7. Frakes, W.B., and Pole, T. An empirical study of representation methods for reusable software components. *IEEE Trans. Softw. Eng. SE20*, 8 (Aug. 1994), 617–630.
8. Humphrey, W. *Managing the Software Process*. Addison-Wesley, Reading, Mass., 1989.
9. McGill, R., Tukey, J.W., and Larson, W.A. Variations of box plots. *Am. Stat. 32* (1978), 12–16.
10. Prieto-Diaz, R. Implementing faceted classification for software reuse. *Commun. ACM 34*, 5 (May 1991), 88–97.
11. Tracz, W. Software reuse myths. In *Software Reuse: Emerging Technology*, W. Tracz, Ed. IEEE Computer Society Press, Washington, D.C., 1988.
12. Tukey, J.W. *Exploratory Data Analysis*. Addison-Wesley, New York, 1977.
13. Welkowitz, J., Ewen, R., and Cohen, J. *Introductory Statistics for the Behavioral Sciences*. 2d ed. Academic Press, New York, 1976.

About the Authors:

WILLIAM B. FRAKES is an associate professor and director of the computer science program at Virginia Tech, and president of the Software Engineering Guild. Previously, he was manager of the Soft-

Continued on page 112