ELSEVIER

The Journal of Systems and Software 57 (2001) 99–106

The Journal of Systems and Software

www.elsevier.com/locate/jss

An industrial study of reuse, quality, and productivity

William B. Frakes [a,*], Giancarlo Succi [b]

[a] Department of Computer Science, Virginia Tech, 7054 Haycock Rd., Falls Church, VA 22043, USA
[b] Department of Electrical and Computer Engineering, University of Calgary, Calgary, Canada

Received 21 October 1999; received in revised form 18 February 2000; accepted 1 May 2000

Abstract

The relationship between amount of reuse, quality, and productivity was studied using four sets of C and C++ modules collected from industrial organizations. The data domains are: text retrieval, user interface, distributed repository, medical records. Reuse in this paper is ad hoc, black box, compositional code reuse. The data generally show that more reuse results in higher quality, but are ambiguous regarding the relationship between amount of reuse and productivity.

Keywords: Software; Reuse; Metrics; Experiment; Empirical; Reuse level; Reuse frequency; Quality; Productivity

1. Introduction

This paper presents a quasi-experimental study of four sets of industrial data to determine the effects of reuse on software quality and productivity. This is an important question since many organizations today are attempting to significantly increase the quality and productivity of their software systems via reuse (Frakes and Isoda, 1994).

Despite its importance, there is a paucity of industrial studies on this topic. Frakes and Terry (1996) provides a survey of these studies. Some recent studies on the relationship between reuse, quality and productivity based on academic data have been reported (Basili et al., 1996; Devanbu et al., 1996). A key difficulty is obtaining industrial data. One serious problem is the type of available industrial data. Industrial data are usually typified by small sample sizes and lack of strict experimental controls. Industrial data also often come in small sets, and conclusions have to be derived from analysis across data sets, rather than within a single set. There are two approaches to this situation. One is to not do empirical, industrial studies until more perfect data become available. As the lack of experimental studies in software engineering attests, this often means that studies are never done because "scientifically acceptable" data never become available.

The other approach is to analyze the data that are available, and learn what we can from them using exploratory quasi-experimental techniques. This is the approach taken in this paper. Our method is to do a correlational study (Campbell and Stanley, 1966) of quality and productivity measures and "amount of reuse" measures derived from four small sets of industrial data.

There are many types of software reuse. A typology of them is reported in Frakes and Terry (1996). The data in this paper are based on ad hoc, black box, compositional code reuse. Ad hoc means that the observed reuse is not part of a repeatable, mandated organizational process. Ad hoc reuse is by far more common than systematic reuse, though the latter is thought to be more powerful. Black box reuse is reuse of a software item without modification. Compositional reuse means that the software system was built by a human programmer out of components, as opposed to generating a system automatically from specifications.

This paper is organized as follows. In Section 2, we define the measures of amount of reuse, quality, and productivity used in this study. Section 3 presents a discussion of the experimental methods used. Section 4 presents the analysis of the four data sets, and Section 5 presents conclusions and suggestions for future work.

* Corresponding author. Tel.: +1-703-538-8374; fax: +1-703-538-8348.

E-mail addresses: wfrakes@vt.edu (W.B. Frakes), giancarlo.succi@enel.ucalgary.ca (G. Succi).

0164-1212/01/$ - see front matter
PII: S0164-1212(00)00121-7

2. Measures used in this study

2.1. Quality and productivity measures

Since the first studies of McCall et al. (1977), it has been clear that it is not possible to measure the quality and productivity of software systems with a single number. Accordingly, many measures of software quality and productivity have been proposed (Conte et al., 1986; Arthur, 1993).

In this paper we use measures of quality and productivity that we have observed to be commonly used in industrial practice. Not all the measures that we use are the best possible measures for describing productivity and quality. Our selection of these measures is constrained by the available data. However, each measure provides a useful perspective on software quality and productivity.

We use the number of non-commentary source lines (NCSLs) as a measure of the size of modules and systems. We compute NCSL for C and C++ by counting in the source file the number of semicolons that are not in comments. Though often criticized, NCSL has been used extensively by practitioners and researchers (Boehm et al., 1995). Number of person days spent developing modules is used as a measure of effort.

The data available to this study for measuring quality are:

- Number of errors per NCSL (Error Density), which is perhaps the most commonly used measure of software quality and has been widely reported for many years (Conte et al., 1986; Humphrey, 1996).
- Number of module deltas (Deltas) where a delta is a change to a software work product such as code. Such a change can be either an enhancement or a repair, and since they have been found to correlate well with faults, deltas are sometimes used for estimating error rates (Drake, 1996). For example, they were used in large switching projects at AT&T since it was a by-product of the formal change control process using tools like SCCS. Several studies use it as a proxy for maintenance effort, such as Li and Henry (1993). Evidence of the relationship between Deltas and Errors is found also in the Developer B data set, which shows a positive correlation between these two variables.
- Quality as perceived by the developers based on the trouble they had in debugging and maintaining the software (Quality Rating). This is a subjective ordinal measure based on an integer scale ranging from 1– worst to 10 – best. Subjective measures of quality have been used in the past in other studies, such as Prechelt and Tichy (1998) and form the basis of survey research, widely used in conjunction with objective measures. Subjective evaluations of advantages or disadvantages of methodologies are also very common, see for instance (Berg et al., 1995; Cline, 1996).

The data available to measure productivity are:

- Number of NCSLs produced per person day (NCSL/Effort). This measure relies on the effectiveness of NCSL as a measure of product size.
- Effort in person days spent per module (Effort/Module) (Basili and Reiter, 1979; Conte et al., 1986). It provides information on how much effort is needed to develop the unit of compilation and deployment. The higher the value of Effort/Module, the lower the productivity.

2.2. Reuse measures

There are several classes of reuse measures (Frakes and Terry, 1996). One of the most important is "amount of reuse" since it indicates how much reuse has taken place in a given software item. We will be using two measures of amount of reuse, reuse level (RL) (Frakes, 1990) and reuse frequency (RF) (Frakes and Terry, 1994). Both of these measures are based on counting the number of *lower level items*, such as functions, used to construct *a higher level item* such as a system.

2.2.1. Reuse level

RL is the ratio of different lower level items reused inside a higher level item over the total number of lower level items used. A reused item can be internal (I) or external (E) based on whether it was developed inside the scope of the project where it is reused. IRL refers to the internal reuse level, while ERL refers to the external reuse level. The RL metric is based on counting item types rather than item tokens.

2.2.2. Reuse frequency

RF measures the number of references to reused items rather than counting items only once, as was done for reuse level. This metric is based on counting item tokens rather than item types. This metric measures the percentage of references to lower level items reused verbatim inside a higher level item versus the total number of references. The definition of IRF and ERF is analogous to that of IRL/ERL. Curry et al. (1999) contains a detailed study on the relationships between RL and RF.

2.3. RL and RF metrics for the C language

To apply RL and RF to the C language, we have to define the concepts of "lower level item" and "higher level item" in a C context. We define these concepts as:

- lower level item: a function,
- higher level item: a file.

The following C code is used to illustrate the calculation of RL and RF for C. `File1.c` and `File2.c` both

reside in the same project. Here we assume that all functions not defined within these two files are external to the project.

```
/* File1.c */
int x, y, z, w;
int f1() {return 1;}
int f2();
int f3();
int g(int, int);
int f(int a, int b) {
  f1();
  f1();
  f2();
  f2();
  g(a, b);
  g(a, b);
  return g(a,b);
}
/* File2.c */
int f2();
int g(int i, int j)
{
  int r = 0;
  f2();
  g(1,2);
  return i+j;
}
```

Table 1 lists the functions within the `File1.c` and `File2.c`, defining each as either internal or external based on their location relative to the project boundary. The last column in the table provides a listing of the number of references made to each function. In the example, although the reference to function `g()` is external to `File1.c` and internal to `File2.c`, it is considered as internal since the function definition resides within the project. However, the functions `f2()` and `f3()` are external to the measured component because they are defined outside the project boundary. Table 2 summarizes the resulting amount-of-reuse metrics. The details of the computations follow.

In `File1.c`, there are a total of three functions or lower level items used within the higher level item or file. In `File1.c` there are calls to one external and two internal functions. This results in an ERL of 1/3. There are seven references to functions: `f1()` (internal) and `f2()` (external) are referenced twice and function `g()` (internal) is referenced three times. Therefore, ERF is 2/7. These measures are computed in the same manner for `File2.c`.

Table 1
Classification of functions for `File1.c` and `File2.c`

File	Function name	Internal/ external	References
File1.c	f1()	Internal	2
File1.c	f2()	External	2
File1.c	f3()	External	0
File1.c	g()	Internal	3
File2.c	f2()	External	1
File2.c	g()	Internal	1

Table 2
Amount-of-reuse in the files of the example

File	ERL	ERF
File1.c	1/3	2/7
File2.c	1/2	1/2

2.4. Reuse level measurement tool – rl

The `rl` software tool (Frakes and Terry, 1994) calculates RL and RF for C code. Given a set of C files, `rl` reports the following information:

1. internal reuse level,
2. external reuse level,
3. internal reuse frequency,
4. external reuse frequency.

The software allows multiple definitions of higher level and lower level abstractions. The allowed higher level abstractions are system, file or function. The lower level abstractions are function or source lines of code.

The RL and RF data for C++ were obtained by first translating the files from C++ to C using the standard AT&T `cfront` utility. Among other tasks, `cfront` produces a one-to-one mapping between methods and functions in C++ and functions in C; the structure of the mapping is well documented. The `rl` tool was then used on the resultant C code, taking advantage of this mapping. It would be better to calculate RLs and RFs directly for C++, but the current version of `rl` does not support this.

3. Structure of the study

We collected four data sets from four different sources. Table 3 summarizes the sizes of the samples and the quality and productivity measures used. An "*X*" in a cell indicates that a given quality or productivity measure was used for a given data set.

This study is an exploratory correlational study (Campbell and Stanley, 1966) of quality and productivity measures and amount of reuse measures. A correlational study is a kind of quasi-experiment. Quasi-experiments are used in settings where strict experimental controls and randomization of treatment conditions are not possible. This is typical of industrial software experimentation. In each data set, we will determine if there is a relationship between each quality and productivity measure available and the two selected amount of reuse measures, ERF and ERL.

Table 3
Quality and productivity measures in this study

Name of the source	Sample size	Error Density	Quality Rating	Deltas	NCSL/Effort	Effort/Module
PRC	3			×		
Developer A	6		×			
Developer B	12	×	×	×	×	×
Software Uno	16				×	×

We will use Spearman's rank correlation (Welkowitz et al., 1990) to determine the presence of a relationship between quality or productivity measures and amount of reuse measures. This correlation statistic can be used for all our data, including ordinal measures, such as Quality Ratings. Spearman's rank correlation is also robust to outliers. Given the small sample size we will not use inferential statistics to test the significance of correlation statistics, but report them only to indicate the magnitude and direction of relationships between variables.

4. Analysis of the data

4.1. PRC Inc

Our first data set was obtained from PRC Inc., a large government contracting company located in Fairfax, VA, USA. The domain of the software was text processing and retrieval. We do not have demographic data on the developers of this software. ERL and ERF were measured for three software modules written in the C language. Deltas in the PRC data were extracted using SCCS, a change management tool on Unix. Table 4 summarizes these measures.

The rank correlation for ERL and Deltas is −0.5 and for ERF and Deltas is −1.0. These data indicate a negative relationship between ERF and ERL, and Deltas. This meets the expectation that more reuse will produce fewer faults, because reusable components are expected to have been tested more thoroughly, or at least to have had longer use allowing errors to be identified and corrected.

4.2. Developer A

These data were collected from a software engineer (Developer A) employed by a medium-sized telecom software company in Europe. The name of the company is omitted for confidentiality. The developer had a B.Sc. in computer science and about two years of experience. The company was certified ISO 9000 and CMM level 2 at the time of the software development. The programming language was AT&T C++, the development environment was Softbench, and the operating system was HP/UX version 9. The target application was a graphical user interface.

Table 4
PRC reuse and module delta data

Name	ERL 0	ERF 0	Deltas
Module 1	0.47	0.4	13
Module 2	0.44	0.44	8
Module 3	0.33	0.25	16

Table 5 presents data from the six software modules we collected from Developer A, and summary statistics of these data. Modules varied in size from 442 to 2543 NCSL with a median value of 2045.5. Quality Ratings varied from 6 to 8 with a median rating of 7. Spearman's rank correlation indicates a positive correlation (0.46) between ERL and Quality Rating, and a stronger positive correlation (0.62) between ERF and Quality Rating. These findings indicate that higher levels of reuse are related to higher perceptions of quality.

4.3. Developer B

These data were collected from another software engineer at the same company as Developer A, but in a different department. Developer B had an M.Sc. degree in Electrical Engineering with five years of industrial experience. These data are based on 12 modules for a distributed reuse library. The working environment of Developer B is the same as Developer A.

The data collected for these modules and summary statistics are given in Table 6. Modules varied in size from 50 to 3197 NCSL with a median value of 679. The

Table 5
Developer A data

Name	NCSL	ERL	ERF	Quality Rating
Module 1	2055	0.44	0.39	8
Module 2	1860	0.41	0.34	7
Module 3	2543	0.46	0.21	7
Module 4	442	0.20	0.19	6
Module 5	2348	0.44	0.27	7
Module 6	2036	0.43	0.24	8
Mean	1880.67	0.40	0.27	7.17
Median	2045.50	0.43	0.26	7
S.D.	745.89	0.10	0.08	0.75
Min	442	0.20	0.19	6
Max	2543	0.46	0.39	8

Table 6
Developer B data

Name	NCSL	ERL	ERF	Errors	Error Density	Deltas	Quality Rating	Effort	NCSL/ Effort
Module 1	478	0.25	0.25	0	0.000	2	7	3	159.33
Module 2	2586	0.30	0.26	0	0.000	3	7	6	431.00
Module 3	662	0.43	0.24	2	0.003	5	8	10	66.20
Module 4	553	0.48	0.42	0	0.000	1	8	2	276.50
Module 5	395	0.23	0.11	3	0.007	3	6	10	39.50
Module 6	2723	0.32	0.18	0	0.000	2	7	7	389.00
Module 7	676	0.33	0.25	1	0.001	3	8	10	67.60
Module 8	3197	0.33	0.22	2	0.000	3	7	15	213.13
Module 9	2516	0.30	0.28	1	0.000	2	7	7	359.43
Module 10	682	0.34	0.20	2	0.002	2	6	5	136.40
Module 11	50	0.43	0.43	0	0.000	1	10	1	50.00
Module 12	2293	0.33	0.31	0	0.000	1	9	3	764.33
Mean	1400.91	0.34	0.26	0.92	0.001	2.33	7.50	6.58	246.04
Median	679.00	0.33	0.25	0.50	0.000	2.00	7.00	6.50	186.23
S.D.	1144.69	0.074	0.09	1.08	0.002	1.15	1.17	4.12	213.83
Min	50.00	0.23	0.10	0	0.000	1.00	6.00	1.00	39.50
Max	3197.00	0.48	0.43	3	0.007	5.00	10.00	15.00	764.33

developer kept a log of the errors he found after releasing the system and the effort it took to develop each module, in person days. The number of errors varied from 0 to 3 with a median value of 0.5. The effort varied from 1 to 15 days with a median value of 6.5. The developer used the Unix utility RCS to track the number of deltas of each module; Deltas varied from 1 to 5 with a median value of 2. Quality Ratings varied from 6 to 10 with a median rating of 7.

Table 7 contains a summary of the rank correlations for Developer B data. There are high negative correlations between ERF and both Error Density and Deltas, indicating that more reuse is related to more correct software. We also found a high positive correlation (0.76) between Quality Rating and ERF indicating that more reuse is related to higher perceptions of quality. The correlations between quality measures and ERL go in the same direction as those of ERF, but are lower.

There is a high negative correlation between ERF and Effort/Module indicating that more reuse is associated with higher productivity for this data set. The correlation between ERL and Effort/Module goes in the same direction but it is lower. The correlations between NCSL/Effort and amount of reuse measures are fairly low and contradictory: the one with ERF is positive and the one with ERL is negative.

4.4. Software Uno

These data were collected at Software Uno S.r.l., a medium-sized software company located in Padova, Italy. The company has around 20 employees. The software was developed between 1991 and 1994. There were three people in the team that developed the software. Two of them had B.Sc. degrees (one in Computer Science and one in Computer Engineering) and each had 5 years of experience at the beginning of the project. The third had an M.Sc. in Electrical Engineering with 8 years of experience at the beginning of the project. The programming language used was Microsoft C under MS/DOS.

These data were collected on 16 files from a system for medical records. The data collected for these modules, and the related descriptive statistics are given in Table 8. Modules varied in size from 46 to 805 NCSL with a median value of 143.5. Software Uno kept a record of the effort spent per module in person days for management purposes.

A positive relationship was observed between ERL and NCSL/Effort (0.31) and between ERF and person days of effort (0.66). Negative correlations were observed between ERF and NCSL/Effort (−0.22) and ERL and person days of effort (−0.45). These results are

Table 7
Spearman's rank correlations for data from Developer B

	Error Density	Deltas	Quality Rating	Effort/Module	NCSL/Effort
ERL	−0.04	−0.18	0.56	−0.22	−0.17
ERF	−0.62	−0.61	0.76	−0.69	0.27

Table 8
Summary of Software Uno data

Name	NCSL	ERL	ERF	Effort	NCSL/Effort
Module 1	98	0.36	0.01	4	24.50
Module 2	304	0.35	0.26	23	13.22
Module 3	145	0.42	0.03	4	36.25
Module 4	327	0.32	0.01	5	65.40
Module 5	120	0.39	0.02	3	40.00
Module 6	111	0.42	0.01	2	55.50
Module 7	46	0.40	0.00	3	15.33
Module 8	805	0.37	0.06	45	17.89
Module 9	749	0.30	0.05	48	15.60
Module 10	105	0.32	0.02	3	35.00
Module 11	113	0.23	0.01	10	11.30
Module 12	248	0.29	0.03	5	49.60
Module 13	142	0.36	0.02	4	35.50
Module 14	197	0.30	0.04	7	28.14
Module 15	106	0.23	0.02	3	35.33
Module 16	467	0.28	0.03	92	5.08
Mean	255.19	0.33	0.04	16.31	30.23
Median	143.50	0.33	0.02	4.50	31.57
S.D.	230.87	0.06	0.06	24.97	17.06
Min	46	0.23	0.00	2	5.08
Max	805	0.42	0.26	92	65.40

contradictory and run counter to what would be expected, since more reuse is generally thought to result in higher productivity, regardless of the measure of amount of reuse. These results are consistent with what we found for Developer B.

5. Conclusions and future work

This paper is an exploratory study of the relationships between amount of reuse, and quality and productivity in software modules from four industrial

Table 9
Summary of the results

Data set	Domain	Data points	Rank correlation	Relationship
PRC	Text retrieval	3		
			ERL vs. Deltas	Negative
			ERF vs. Deltas	Negative
Developer A	User interface	6		
			ERL vs. Quality Ratings	Positive
			ERF vs. Quality Ratings	Positive
Developer B	Distributed repository	12		
			ERL vs. Quality Ratings	Positive
			ERF vs. Quality Ratings	Positive
			ERL vs. Error Density	Marginally negative
			ERF vs. Error Density	Negative
			ERL vs. Deltas	Marginally negative
			ERF vs. Deltas	Negative
			ERL vs. NCSL/Effort	Marginally negative
			ERF vs. NCSL/Effort	Marginally positive
			ERL vs. Effort/Module	Marginally negative
			ERF vs. Effort/Module	Negative
Software Uno	Medical records	16		
			ERL vs. NCSL/Effort	Positive
			ERF vs. NCSL/Effort	Negative
			ERL vs. Effort/Module	Negative
			ERF vs. Effort/Module	Positive

organizations. The data sets are from four different domains: text retrieval, user interface, distributed repository, and medical records. The data in this paper are based on ad hoc, black box, compositional code reuse. The modules in the study were written in C and C++.

The amount of reuse was measured with RL and RF. The `rl` tool was used to calculate these data. Quality was measured subjectively by asking the developers to rate the modules they developed on a 10-point scale, and with Error Densities and module deltas. Productivity was measured with Effort per Module and NCSL/Effort. Spearman's rank correlations were calculated to summarize the direction and magnitude of the relationships between pairs of variables. The results are summarized in Table 9.

The data generally support the conclusion that more reuse will result in improved quality. In the PRC data there is a negative relationship between ERL, ERF, and Deltas. In the Developer A data, there is a positive relationship between ERL, ERF, and Quality Ratings. In the Developer B data, there is a negative relationship between ERF, ERL, and Deltas and between ERF and Error Density. There is a positive relationship between ERL, ERF, and Quality Ratings.

The data are inconsistent regarding the relationship between reuse and productivity with some relationships being positive and others negative. This inconsistency warrants further study.

This study, and the few others in this area, should indicate to software practitioners that increases in software quality and productivity from reuse cannot be taken for granted. The best approach for such organizations will be to collect data such as we have reported in this study for their own organizations, and use these data to guide their reuse projects.

Future research in this area is certainly warranted. The key is more and better data collected from widely representative industrial software organizations. True experimental studies with strict controls and replications are also needed, though industrial resource constraints and concerns for privacy will continue to be barriers. In the meantime, it is important to keep collecting and analyzing available industrial data.

Acknowledgements

We would like to thank Prof. G. Kovacs for his insightful comments, N. Scioscia and G. Cardino for their help in finding data, CIM-EXP (Budapest, Hungary), PRC (Falls Church, VA), and Software Uno S.r.l. (Padova, Italy) for the data they provided us. Thanks also to Omar Alonso, Gabriella Belli, Gloria Hasslacher, Eric Liu, Romana Spasojevic, Milorad Stefanovic, Raymond Wong, and Jason Yip for their reviews of a draft of the paper. The second author thanks the University of Calgary, the Government of Alberta, the Canadian National Science and Engineering Research Council for partially supporting this research.

References

Arthur, L.J., 1993. Improving Software Quality – An Insider's Guide to TQM. Wiley, New York.

Basili, V.R., Reiter, R.W., 1979. An investigation of human factors in software development. IEEE Computer 12 (12), 21–38.

Basili, V., Briand, L., Melo, W., 1996. How reuse influences productivity in object-oriented systems. Communications of the ACM 39 (10), 104–116.

Berg, W., Cline, M.P., Girou, M., 1995. Lessons learned from the OS/400 OO project. Communications of the ACM 38 (10), 54–64.

Boehm, B., Clark, B., Horowitz, E., Westland, C., Madachy, R., Selby, R., 1995. Cost models for future software life cycle processes: COCOMO 2.0. Annals of Software Engineering 1 (1), 57–94.

Campbell, D.T., Stanley, J.C., 1966. Experimental and Quasi-Experimental Designs for Research. Rand Mc.Nally College Publishing Company, Chicago, IL.

Cline, M.P., 1996. The pros and cons of adopting and applying design patterns in the real world. Communications of the ACM 39 (10), 47–49.

Conte, S.D., Dunsmore, H.E., Shen, V.Y., 1986. Software Engineering Metrics and Models. Benjamin/Cummings, Menlo Park, CA.

Curry, W., Succi, G., Smith, M., Liu, E., Wong, R., 1999. Empirical analysis of the correlation between amount-of-reuse metrics in the C programming language. In: Proceedings of the 1999 Symposium on Software Reusability. ACM Press, New York, pp. 35–140.

Devanbu, P., Karstu, S., Melo, W., Thomas, W., 1996. Analytical and empirical evaluation of software reuse metrics. In: Proceedings of the 18th International Conference on Software Engineering. ACM Press, New York, NY, pp. 189–199.

Drake, T., 1996. Measuring software quality: a case study. IEEE Computer 29 (11), 78–87.

Frakes, W., 1990. An empirical framework for software reuse research. In: Proceedings of the Third Annual Workshop on Software Reuse (Technical Report No. 9014), Syracuse University CASE Center.

Frakes, W., Isoda, S., 1994. Success factors of systematic reuse. IEEE Software 11 (5), 14–19.

Frakes, W., Terry, C., 1994. Reuse level metrics. In: Proceedings of the Third International Conference on Software Reuse: Advances in Software Reuse. IEEE CS Press, Los Alimitos, CA, pp. 139–148.

Frakes, W., Terry, C., 1996. Software reuse and reusability models and metrics. ACM Computing Surveys 28 (2), 415–435.

Humphrey, W., 1996. Introduction to Personal Software Process. Addison-Wesley, Reading, MA.

Li, W., Henry, S., 1993. Object oriented metrics that predict maintainability. Journal of Systems and Software 23 (2), 111–122.

McCall, J., Richards, P., Walters, G., 1977. Factors in Software Quality, NTIS AD-A049-014, 015, 055.

Prechelt, L., Tichy, W.F., 1998. A controlled experiment to assess the benefits of procedure argument type checking. IEEE Transactions on Software Engineering 24 (4), 302–312.

Welkowitz, J., Ewen, R.B., Cohen, J., 1990. Introductory Statistics for the Behavioral Sciences/Includes Software, fourth. Academic Press, New York.

William Frakes is an associate professor in the Computer Science Department at Virginia Tech. He chairs the IEEE TCSE committee on software reuse, and edits ReNews. He has a B.L.S. from the University of Louisville, an M.S. from the University of Illinois at Urbana-Champaign, and an M.S. and Ph.D. from the Syracuse University.

Giancarlo Succi is a Professor of Electrical and Computer Engineering at the University of Alberta, Edmonton, AB. He holds a Laurea degree in Electrical Engineering (Genova, Italy, 1988) Italy, an M.Sc. in Computer Science (SUNY Buffalo, NY, 1991), and a Ph.D. in Electrical and Computer Engineering (Genova, Italy, 1993). His research interests include software reuse, software metrics, software product lines, and software engineering economics. He has chaired and co-chaired several international, scientific events and published papers in international journals and conferences.