Strategies for Software Reuse: A Principal Component Analysis of Reuse Practices

Marcus A. Rothenberger, Kevin J. Dooley, Uday R. Kulkarni, *Member*, *IEEE*, and Nader Nada

Abstract—This research investigates the premise that the likelihood of success of software reuse efforts may vary with the reuse strategy employed and, hence, potential reuse adopters must be able to understand reuse strategy alternatives and their implications. We use survey data collected from 71 software development groups to empirically develop a set of six dimensions that describe the practices employed in reuse programs. The study investigates the patterns in which these practices co-occur in the real world, demonstrating that the dimensions cluster into five distinct reuse strategies, each with a different potential for reuse success. The findings provide a means to classify reuse settings and assess their potential for success.

Index Terms—Reusability, systematic software reuse, software process improvement, quality, reuse success, reuse classification scheme, best practices.

1 Introduction

Systematic software reuse is a technique that is employed to address the need for improvement of software development efficiency and quality [49]. This involves the use of artifacts from existing systems to build new ones in order to improve quality and maintainability and to reduce cost and development time [1]. This study explores practices employed for code reuse. These reuse practices vary by extent of organizational support, integration in the overall development process, and techniques employed. Depending on the reuse practice, reuse may happen in an unplanned fashion when individual developers recognize reuse opportunities informally or it may be planned for by building and maintaining a software repository that supports the predictable and timely retrieval of reusable artifacts [22].

Writing reusable software requires additional development effort compared to writing code in a nonreuse setting. Reusable components need to be developed in a generic fashion that allows their use in various contexts. Creating a software repository and populating it with reusable components is a substantial investment for an organization that can only pay off in the long run when development effort is saved through reuse in software projects [4]. Hence, the adoption of a software reuse program involves risk and the choice of a reuse strategy can be crucial to its success. Although Morisio et al. [41] have argued that there may be more than one successful approach to reuse, the adoption of different practices that can facilitate reuse may provide different results, thus organizations must make an informed decision concerning whether or not to implement software reuse and, if so, how.

The literature has discussed broad reuse classification schemes [9], [33], [50] that distinguish between various approaches to reuse. In their broad definitions of reuse, these publications include the reuse of everything associated with a software project, such as procedures, knowledge, documentation, architectures, design, and code. Prieto-Díaz's [50] classification scheme characterizes different approaches to reuse along a variety of dimensions: substance, scope, mode, technique, intention, and product. His model considers reuse in the widest sense by including facets ranging from automatic code generation to reuse of ideas. As a consequence, its conceptualization of practices is at a very abstract level. In order to engage the reuse phenomenon at a level closer to the organizational reuse setting, we are focusing on the systematic reuse of software code components. The reuse of other artifacts, such as ideas, analysis documents, test cases, designs, or the automatic generation of code are beyond the scope of this research.

Existing code reuse literature has identified reuse practices and success factors using case studies and surveys. A major reuse effort reported in the literature is the REBOOT (Reuse Based on Object-Oriented Techniques) consortium [61]. This effort was one of the early reuse programs that recognized the importance of not only the technical, but also the organizational aspects of reuse. Kim and Stohr [31] confirmed this by arguing that software reuse could only succeed if also nontechnical issues are considered. Edwards [16] reports on the results of a survey of reuse experts which also indicates that additional work is required with respect to the nontechnical reuse issues. Lee and Litecky [35] have examined success drivers among organizations that took an object-based approach to reuse with the programming language Ada. The findings of this study highlight the importance of factors such as management support and domain knowledge. Several other studies

- *M.A. Rothenberger is with the School of Business Administration, University of Wisconsin-Milwaukee, PO Box 742, Milwaukee, WI 53201. E-mail: rothenb@uwm.edu.*
- *K.J. Dooley and U.R. Kulkarni are with the W.P. Carey School of Business, Arizona State University, Tempe, AZ 85287. E-mail: {kevin.dooley, uday.kulkarni}@asu.edu.*
- *N. Nada is with the Department of Computer Science, Sharjah College, Sharjah, United Arab Emirates. E-mail: nader@shjcollege1.ac.ae.*

Manuscript received 17 Apr. 2002; revised 17 June 2003; accepted 24 June 2003.
Recommended for acceptance by A. Mili.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 116367.

also stress the importance of management support [1], [23], [29], [53]; others mention the integration of reuse into the development process [12], [29]. Biggerstaff [8] introduces infrastructure considerations; Frakes and Fox [21] stress the importance of training for reuse, among other factors. Frakes and Isoda [22] and Banker et al. [5] emphasize reuse measurement. This brief review demonstrates that prior research presents reuse success factors without formally considering interactions between them. However, in the real world, best practices as described by such success factors are not independently implemented. Instead, they occur in the form of archetypal sets of practices. As a consequence, to truly help organizations to implement a reuse program, we shall not focus on separate reuse success factors, but we shall examine in what combinations practices co-occur and how such archetypal sets of practices affect reuse success. We have drawn from prior literature to identify specific practices associated with systematic software reuse, including *reuse measurement*, the degree of *commonality of the architecture*, and the level of *management support* [1], [22], [32], [35]. The present research uses these literature-identified reuse practices to develop a classification scheme for software reuse.

It is important to understand how these reuse practices can be aggregated into constructs that describe the differences between systematic reuse strategies. Such insights can help organizations considering a reuse program in deciding how to go about systematically adopting software reuse. Our study uses survey data from software development groups working with software reuse to reduce the numerous reuse success factors identified in prior studies to a concise set of six reuse dimensions by grouping co-occurring practices into single constructs. Thus, we are able to describe reuse settings with a vector of only six dimensions. Since reuse strategies do not occur in all possible permutations of the vector values, we find that they cluster into five distinct strategies with different levels of reuse success. The contributions of this research are the identification of the reuse dimensions and their theoretical development, as well as the development of reuse strategy clusters, as observed in the real world. The study also yields a qualitative investigation on the effect of reuse strategies on the success of a reuse program.

The paper is structured as follows: Section 2 presents the systematic reuse practices from the literature that serve as a basis for the survey questions. The section also briefly describes the data collection and the statistical methods used for data analysis. Section 3 presents the discussion of software reuse dimensions "discovered" by our analysis. Section 4 investigates the implementation clusters and their linkage to reuse success. The conclusion summarizes the results of this study pointing to future research directions.

2 Research Approach

The term "*reuse dimension*" is used to denote a construct composed of multiple reuse-related practices that an organization may employ. Consolidation of various reuse practices into dimensions allows a high-level view of the various reuse variables that may be participating in a reuse strategy. Our motivation in this research is based on the belief that such dimensions can be proactively used in formulating a well-thought-out reuse strategy. Hence, the objective of this research is to determine the dimensions that constitute systematic software reuse and evaluate the impact of these dimensions on actual reuse success.

2.1 Reuse Practices

As a first step, we synthesized existing software reuse research literature in order to determine the dominant themes apparent in discussions about reuse. From this, we identified a number of reuse practice variables for inclusion in our survey instrument.

While survey data is most often used for theory testing with firm theoretical notions already in hand from prior research, in our research, the survey data was used to enhance the theoretical constructs by allowing the survey data to suggest the final set of reuse dimensions [14]. The items included in the survey instrument along with its applicable literature reference are presented in Table 1. Responses were collected as interval measures on a five-point scale. Table 1 shows the survey items grouped by reuse practice categories. The practice categories give a sense of the survey items in aggregate. The aggregates are derived by composing an affinity diagram [18], a technique that collates textual data into clusters via an iterative, manual process performed by domain experts (in this case, the authors). The survey items clustered into the following reuse practice categories: project similarity [22], [27], reuse planning [22], measurement [48], process improvement [21], formalized process [21], management support [1], education [21], object technologies [43], and commonality of architecture [22].

2.2 Data Collection

Comprehensive lists of organizations that develop software are publicly available and easy to access, but there is no national or international software engineering database or census bureau representing a population of software developers who practice software reuse. Software reuse is still not widely adopted; only a small number of development groups currently use the reuse paradigm.

We used the following approach to reach the sample: First, potential survey respondents were identified from conferences/symposia/workshops that extensively cover the topic of software reuse. Further, subscribers of various software reuse groups were identified. Next, the survey instrument, augmented with demographic questions such as title, group, and company size, was posted on a web site for easy access by software developers. After that, the subjects were sent e-mail messages with a request to participate in the survey. The list of conferences and subscription groups whose participants/subscribers were contacted is given below:

Conferences:

- The International Conference on Software Engineering,
- The Symposium on Software Reusability, and
- The European Reuse Workshop.

TABLE 1
Survey Items

Reuse Practice Category [Ref.]	Survey Question(s)
Project Similarity [22], [27], [51], [52], [54]	1. During (after) the project requirements phase we realized high commonality with the requirements of previous project(s). 2. During (after) the project design phase we realized high commonality with the design of previous project(s). 3. During (after) the project coding phase we realized high commonality with the code (documentation) of previous projects.
Reuse Planning [21], [22], [33], [34], [38], [54], [62]	4. We have done thorough domain analysis of our products. 5. We have an efficient requirement analysis tool, which links our most common end-user requirements with the reusable software components, which satisfy them. 6. Our software development process is built around a core set of reusable software components as the foundation for our products.
Measurement [5], [22], [32], [48]	7. Performance of the software reuse process is carefully measured and analyzed to identify weaknesses, and plans are established to address the weakness.
Process Improvement [21]	8. Pilot projects were used effectively to refine software reuse "best practices" prior to adopting them for routine use. 9. Projects document their software reuse lessons learned, which in turn are used to improve the organization's software reuse process.
Formalized Process [21], [29]	10. Software developers and maintainers precisely follow a software reuse process, which is defined and integrated with the organization's software development process. 11. We have a valuable process for certifying reused software components 12. Our configuration management system does an exceptional job in keeping track of the projects, which use each reusable software component.
Management Support [1], [23], [29], [32], [35], [53]	13. Senior management has demonstrated a strong support for software reuse by allocating funds and manpower over a number of years. 14. There is a strong influential individual(s) in senior management who actually advocates and supports developing a software reuse capability.
Education [21] [32]	15. Our staff has a very competent software reuse education and/or can attend training courses (internal and/or commercial).
Object Technologies [32] [43]	Technologies used on the project 16. Object-oriented Programming Languages 17. Distributed Object Computing 18. OMT/UML Domain Modeling Languages 19. Either Ada, C++, Java, Eiffel or Smalltalk
Commonality of Architecture [22], [32]	20. We rely heavily on a common architecture across our product line.

Subscription Groups:

- Software Reuse Factors (Sonnemann's list, 109 subscribers),
- NASA Software Reuse Workshop (Patterson's list, 200 subscribers),
- Reusable Software Components Resources (Levine's list, 50 subscribers),
- IEEE Software Reuse Group (90 subscribers), and
- Software Productivity Consortium members (200 subscribers).

Seventy-seven participating development groups belonging to 67 different organizations responded to the survey. Among the survey participants, six respondents omitted more than 15 percent of the questions on their respective questionnaires. Leaving a large number of survey items unanswered may indicate that there is a poor reliability of the responses. For that reason, we excluded these six respondents and used the remaining 71 responses for the analysis. The respondents included project managers, software engineers, developers, software development consultants, and software engineering researchers in profit and nonprofit organizations in a variety of industries, including telecommunication, financial services, avionics, government, and education. There were no duplicate participants within a development group and almost 80 percent of the participants were working in organizations with more than 200 employees.

Finally, we dealt with missing values and the possibility of nonresponse bias in a generally accepted manner as follows: When missing values are under a threshold of

15 percent of the total number of cases, one can substitute them with the means of the responses [42]. In our case, the percentage of omitted responses was well below the threshold; hence, missing values were replaced with the means. A well-accepted method of testing nonresponse bias is to look for significant differences between early and late responses [2], especially when the actual size of the population is unknown. A t-test comparing the early and the late quartile responses exhibited no significant difference. This result suggests that no effect must be attributed to the potential difference between respondents and nonrespondents.

2.3 Data Analysis

When practices cooccur in a systematic pattern of some sort, one may interpret this at least as an association emergent from management action. For example, if the two items pertaining to *requirements analysis tools* and *domain analysis* are (positively) correlated at a magnitude suggesting statistical significance, it indicates that the co-occurrence of these practices may be intentional or it may stem from a larger program of practices being implemented systemically.

We used principal component analysis (PCA, also known as factor analysis) to identify these patterns of co-occurring practices throughout the entire data set. PCA aggregates these correlations between reuse practices into a correlation matrix. PCA then performs a rotation of this matrix such that all eigenvectors are orthogonal to one another. The eigenvectors represent the new "dimensions" that explain the correlation structure in the data and the "loadings" for each item within a given eigenvector indicate the magnitude with which that item is weighed within the identified dimension [14]. These dimensions are then labeled according to the researchers' examination of the items contained within each dimension. In order to reduce bias in this labeling, each of the three researchers came up with independent labels and then met to obtain a consensus decision. The reader can also examine the labels relative to the items to determine the level of face validity that this labeling has achieved.

The results of the PCA and the labeling and meaning of the dimensions are presented in the following section. The next analysis task is to establish the connections between the dimensions of reuse and the success of reuse initiatives. Established measures of reuse success such as those discussed in the literature [7], [24], [36], [57] were used for this part of the analysis. This part of the analysis is the basis for establishing how reuse dimensions can be used to formulate a good reuse strategy. The dimensions resulting from the PCA and the development of propositions describing the dimensions' effects on reuse success are presented in the following section.

3 Dimensions of Systematic Software Reuse Strategies

Six different dimensions are identified (components that have an eigenvalue greater than 1.0 are considered significant [14]), and these six dimensions collectively account for two-thirds of the overall variance, which is considered quite good. Specific items (and, thus, practices) are associated with the six dimensions by examining the factor loadings. An item is considered part of a dimension (factor) when its loading is above 0.40; this is the magnitude that is used to determine whether the factor loading is statistically significant [14]. This leads to a fairly unambiguous assignment of items to dimensions, with the exception of items 10, 12, and 15, which had factor loadings of over 0.40 for two factors. These items were assigned to the dimension in which their loading was highest, as is customarily done. In the cases of questions 10 and 12, the highest loading was into the *Formalized Process* factor, but the loadings of these items into the *Planning and Improvement* factor was only slightly lower. Our decision to include the two items in the *Formalized Process* factor carries face validity since both items ("following a detailed process" and "logging reuse occurrences") describe the presence of defined best practices as captured by the *Formalized Process* construct. Similarly, question 15's highest loading was into the *Management Support* factor, while it loaded almost as high into *Planning and Improvement*. The decision to assign the item ("availability of reuse education and training") to *Management Support* was carried by the argument that only management who is supportive of reuse will make the resources available that are required for reuse education and training. The results are shown in Table 2.

Examining the *reliability* of the collection of items constituting these dimensions enables assessment of their integrity. A collection of items is considered reliable if they have strong patterns of covariance, implying that they are measuring the same phenomenon. Table 2 shows the most common measure of the reliability of a set of items, Cronbach's alpha, for each of the six dimensions. The measures range from 0.71 to 0.87. According to heuristics frequently employed for the interpretation of Cronbach's alpha, a value of 0.7 is considered acceptable [42].

In the following sections, we describe each of the six dimensions resulting from the factor analysis. We discuss how the theory-based reuse practices were empirically combined into six reuse dimensions. Each paragraph concludes with a proposition that is based on the relationship between the presence (or implementation) of these reuse dimensions and the level of success the organization realizes in its reuse effort. *Success* is defined broadly here and constitutes both the effectiveness and timeliness of the development project, as well as the quality and cost of the final product.

3.1 Planning and Improvement

The empirical analysis indicates that the theory-based practices of the categories reuse planning, measurement, and process improvement are describing one underlying reuse dimension, which we shall name *planning and improvement*. This dimension relates to the quality of the reuse process. The three theory-based categories, reuse planning, measurement, and process improvement, are all constructs indicating the extent of quality control exercised over the reuse process. The items are shown below.

TABLE 2
Factor Loading

	Mean (Question)	Std. Deviation (Question)	Factors: Planning & Improvement	Project Similarity	Object Technologies	Management Support	Formalized Process	Common Architecture
Question								
8	3.01	1.30	**.72531**	-.02217	-.15291	.12648	.06756	.04555
9	2.97	1.20	**.72474**	.16033	-.10733	.32120	-.18944	.16568
4	3.00	1.23	**.61777**	-.04877	.03109	.04198	.21329	.50135
5	2.54	1.31	**.60590**	-.13861	.10773	.00138	.13711	-.08190
7	2.55	1.19	**.59296**	.01762	-.18424	.33573	.31181	-.22101
6	3.62	1.18	**.50442**	.09259	-.23422	.25669	.20444	.11184
2	3.37	1.01	-.11621	**.93252**	.05895	-.06540	.10248	.16797
1	3.46	1.06	-.01134	**.88345**	.01553	-.14160	.13386	.05675
3	3.26	1.02	.10339	**.83379**	.06976	.22364	-.07429	-.03682
16	2.86	2.01	-.10055	.12245	**.84257**	.05317	.01486	-.16493
19	3.54	1.94	-.03392	.05869	**.75499**	.06515	.01914	-.11246
17	1.79	1.60	-.17954	-.12122	**.65290**	-.12064	-.00478	.03275
18	1.90	1.68	.08181	.11129	**.63465**	-.10652	-.06694	.33900
14	3.46	1.20	.18101	-.08321	.09280	**.86481**	.09127	.13459
13	3.06	1.21	.23798	.03906	-.08390	**.79639**	.27436	.11964
15	2.83	1.38	.43799	.05456	-.12109	**.44825**	-.02634	.00675
11	2.83	1.20	-.01739	.17140	-.03627	.16142	**.85845**	.17215
10	2.94	1.12	.50060	.11256	-.09750	.23584	**.61110**	-.06145
12	2.86	1.20	.48061	-.10829	.13698	.02263	**.60677**	.02384
20	3.74	.97	.00484	.15979	-.07692	.22576	.09019	**.80716**
Mean (Factor)			2.95	.3.36	2.52	3.12	2.88	3.74
Std.Dev. (Factor)			.85	.92	1.34	1.02	.93	.97
Eigenvalue			4.985	2.726	2.086	1.286	1.186	1.066
% Var Expl.			24.9	13.6	10.4	6.4	5.9	5.3
Cumulative % Var Expl.			24.9	38.6	49.0	55.4	61.3	66.7
Chronbach's Alpha			$\alpha = .78$	$\alpha = .87$	$\alpha = .72$	$\alpha = .72$	$\alpha = .71$	N/A

Theory-Based Category: Reuse Planning

- Question 4: Reuse Planning: Domain analysis.
- Question 5: Requirements linked to reusable software components.
- Question 6: Process built around a core set of reusable components.

Theory-Based Category: Measurement

- Question 7: Reuse performance measurement.

Theory-Based Category: Process Improvement

- Question 8: Refinement of "best practices" through pilot projects.
- Question 9: Lessons learned documented to improve reuse process.

Software reuse is best thought of as a process and, like any organizational process, it can greatly benefit from *planning and improvement*. Effective planning enables the organization to identify what the requirements of the product are and plan ahead of time where opportunities for reuse can be realized. Requirements may also be negotiated as a slight change in them may bring about increased opportunity for reuse and, thus, reduced project time and cost. These trade offs between commonality and distinctiveness are best made up-front in the planning process [55]. Empirical studies of new product development in general show that a strong link between customer needs and the development process is beneficial to product and project outcomes [10]. A study by the General Accounting Office [65] of commercial development practices found that "disciplined paths" must exist in order to infuse the

technological capabilities of the firm (e.g., reusable software modules) into the specifications of a new product.

Continuous improvement is also an essential part of a quality approach to managing the reuse process. This requires measuring reuse outcomes and providing feedback to software developers and managers so that they can better understand how to improve performance [30]. Only then can problems be quickly identified and the reuse program be modified accordingly. Effective reuse measures must account for the various tasks involved in reuse, such as production, retrieval, and integration of components [58]. The selection of performance measures depends on individual management goals [45], thus establishing priorities for developers [13].

When process improvements are identified, it is often suggested that they be tried on a small scale first, as a full-scale implementation is costly and may not yield the results intended. Eisenhardt and Tabrizi [17] suggest that development processes be thought of as rapid prototyping laboratories, where small-scale experiments facilitate rapid and exploratory learning. Lessons learned are then documented so that future projects can benefit from the learning that occurred within previous projects. From this, the following proposition is developed.

P_1: Higher levels of *planning and improvement* are associated with higher levels of reuse program success.

3.2 Formalized Process

The empirical results validate the theory-based category of a *formalized process*. The items are shown below.

Theory-Based Category: Formalized Process

- Question 10: Developers follow a detailed process.
- Question 11: Component certification process.
- Question 12: Logging reuse occurrences (each project/component pair).

Having a *formalized process* increases the chance that the project success can be repeated. A formal process facilitates adherence to the established best practices, standardization of practices across multiple projects (which enhances learning), and helps less-expert developers to succeed via reliance on a standard process. As projects follow a formal process, responsibilities are made clear, and the reliance of one function's work on another—for example, between coding and testing—is understood and points of review and feedback are specified. Formal processes help workers focus on the collective aspects of their work rather than their functional responsibilities and help project personnel to decide what to work on and when [64]. If projects follow a formal process, variance between projects is likely to be reduced and, thus, outcomes will become more predictable. A formalized process embeds the mechanistic decisions that a developer faces into a standardized routine so that developers can spend more time on the creative aspects of their work. Griffin [26], in a review of studies on new product development, found that numerous studies supported the proposition that a formal, structured development process benefited project outcomes, especially when projects were complex. A formalized reuse process must also ensure the repeatability of the reuse success by enforcing component standards. A required certification process can guarantee that every part of the repository meets the desired performance standards. Poulin has reported IBM's positive experience with a reuse program that uses such a certification process [47]. In the domain of software development, a formalized, structured process is the basic goal of the first three levels of the capability maturity model (CMM) [44] and has been found to have a positive benefit on software development project outcomes [25], [28]. This leads to the following proposition.

P_2: Higher levels of *formalized process* are associated with higher levels of reuse program success.

3.3 Management Support

The results of the factor analysis suggest that both theory-based notions of management support and education describe one reuse dimension that we shall name *management support.* The discussion below will show that availability of reuse education and training can be an indicator of management's support for reuse. The items are shown below.

Theory-Based Category: Management Support

- Question 13: Senior management support.
- Question 14: Reuse champion in management.

Theory-Based Category: Education

- Reuse education and training provided for staff.

Management support is often considered one of the most basic prerequisites for effective organizational practice, as it embodies leadership [66], decision making [40], and organizing resources [15]. First, management support consists of allocating resources (funds, manpower) for reuse over an extended period of time. Resources may be based on physical capital (facilities, infrastructure), human capital (people and skills), or organizational capital (rules, routines, rewards) [6]. By selecting which resources to provide, senior management enacts a strategic-level decision about which resources are valuable and important relative to competitive advantage [46]. Resources provide the energy for organizational attention and action; stifling resources in a particular area not only creates an obvious shortfall of attention, but also sends a signal to employees that the area is not significant or important. In this manner, resources tend to beget more resources and further potential emerges from within. Tangible resources tend to be more important when the market environment is relatively stable and intangible resources (such as championing and training) tend to be more important when the market environment is relatively instable [39]. Management support can facilitate the specific allocation of resources towards education and training. Thus, the availability of reuse education and training is an indicator for the degree of management support that reuse enjoys in an organization.

Second, management support may come in the form of a champion. Champions are vocal advocates of a particular issue (i.e., reuse) and ensure that the issue is considered systematically in organizational decisions. A champion ensures that top management not only considers the issue rationally, but also politically. Champions can also take on

the role of a change agent, attempting to allocate resources and change policy and procedure so that new practices can become embedded in organizational routines [56]. From the above, the following proposition is developed:

P_3: High levels of *management support* are associated with high levels of reuse program success.

3.4 Project Similarity

The empirical results validate the theory-based category of *project similarity*. This dimension assesses how alike projects within one development group are with respect to requirements, design, and implementation. The items are shown below:

Theory-Based Category: Project Similarity

- Question 1: Commonality of requirements across projects.
- Question 2: Commonality of design across projects.
- Question 3: Commonality of code across projects.

Projects can be similar by design or by accident. Projects may be similar if an organization develops products for a limited range of customers or constrains the types of projects it works on. The linkage between project similarity and the opportunity for reuse is obvious—in the extreme, if a new project is identical, then full reuse can be achieved (as is the case in reselling off-the-shelf products to multiple customers). Having projects that are too similar though may indicate a too narrow organizational focus and may decrease the adaptive capacity of the firm.

Specific to the domain of software reuse, concentrating on the development of closely related domain-specific applications allows putting together a core set of highly useful reusable components within the domain [27]. For example, business process and supply-chain systems for a particular industry are more likely to incur reuse opportunities than the same systems across different industries. Developers who work within a well-defined domain can leverage their experience across the reuse projects with regards to reuse opportunities and knowledge of the repository [32]. The proposition reflects the above.

P_4: Higher levels of *project similarity* are associated with higher levels of success of the reuse program.

3.5 Object Technologies

The factor analysis supports the theory-based category of *object technologies.* This dimension captures the extent of object technology used on reuse projects. This factor includes the use of object-oriented programming languages, the use of distributed object computing techniques, and the use of object-oriented domain modeling languages. The items are shown below.

Theory-Based Category: Object Technologies

- Question 16: Use of object-oriented programming languages.
- Question 17: Use of Distributed object computing.
- Question 18: Use of OMT/UML Domain Modeling Languages.
- Question 19: Use of either Ada, C++, Java, Eiffel or Smalltalk.

Object-orientation is an increasingly popular approach to facilitate reuse. Methodologies cover all development phases, from domain analysis to programming. Hence, many researchers see the object-oriented paradigm as the technique of the future for reuse [36], [50]. Object-orientation provides a set of techniques that are conducive of the multiple uses of software artifacts across projects. Encapsulation forces the developers to clearly define the interfaces of each class to the outside world. The practice of defining data structures and code in the same class is keeping the elements that need to be reused as a unit within one framework. Object-oriented analysis and design forces the developers to think of the problem in terms of the businesses process that can provide opportunities for reuse. As a consequence, the following proposition is developed.

P_5: Higher levels of *object technologies* are associated with higher levels of reuse program success.

3.6 Common Architecture

The factor analysis has shown that the question based on the theory notion of *common architecture* did not load with any other concept. Thus, this item will form its own dimension.

Theory-Based Category: Common Architecture

- Question 20: Common architecture across product line.

Specific to software reuse, *common architecture* can be viewed as a form of reuse itself, as well as a means to facilitate the reuse of particular software modules. A common architecture enables the formation of product platforms—sets of similar products that are developed rapidly from a common starting point [37], [63]. A common architecture may provide a solution to growing product complexity [3] and may also aid mass-customization and postponement [19].

Organizations that develop a common architecture analyze application domains to identify generic designs that can be used as templates for integrating reusable parts [50]. The reuse of an architecture forces the definition of reuse standards [11]. Standards such as common interfaces, component size, ways of specifying input/output, and documentation are among the key factors to reuse success [52]. Often, such standards are operationalized as design templates that can support the identification of suitable reusable components, thus reducing time spent on retrieval [59]. A *common architecture* defines the rules for developing components by defining interfaces and data formats. These standards reduce the effort spent on customizing component-based software [22]. Based on this, the following proposition is developed.

P_6: Higher levels of *common architecture* are associated with higher levels of reuse program success.

TABLE 3
Dependent Variables

Dependent Variable	Ref.	Survey Question(s)
Reuse Benefit	[7] [36] [48] [57]	We have seen significant improvement in our primary motives for implementing software reuse.
Strategic Impact	[22] [48]	Management has created new business opportunities that take advantage of the organization's software reuse capability and reusable assets. Our decision to bid on new projects or enter new markets are based heavily on our software reuse capabilities.
Software Quality	[24] [36] [48]	Our reusable software components have proven through operational use to be highly reliable.

4 Associations between Reuse Dimensions and Success

In this section, we present five archetypal software reuse settings as they occur in the real world. We will discuss to what extend these strategies are indicative of reuse success as it is measured using three variables derived from the reuse literature.

4.1 Measures for Reuse Success

Promises and potential benefits of systematic software reuse that have been discussed in the literature were grouped into three conceptual issues that measure the success of a software reuse program. These conceptual issues are three dependent variables, each measuring a different aspect of reuse success in the context of the subsequent analysis. *Reuse benefit* focuses on the operational benefit of reuse, measuring to what extent the organization obtains the benefit that was desired from reuse; s*trategic impact* goes a step further by measuring the ability of management to translate operational reuse benefits into organizational benefits; *software quality* measures the improvement to the error rate that is a consequence of reuse achieved, whether it is systematic or accidental reuse; thus, it may indicate success even if the other two measures do not. It has been confirmed that the multiple items in *Strategic Impact* are significantly correlated and load together in one factor (Cronbach's Alpha = 0.75). Because of the single item nature of the other two success measures, no Cronbach's Alpha could be calculated. The items are shown in Table 3.

Reuse Benefit. When organizations invest in making reuse part of their software development process, they expect to obtain a number of benefits. Potential benefits include a *reduction in development effort and cost* [7], *faster time-tomarket* [24], and *decreased maintenance effort* [57]. *Reuse benefit* measures to what extent expected benefits have materialized after the adoption of software reuse.

Strategic Impact. An important goal of a major change such as the adoption of a reuse methodology is the realization of new business opportunities. *Strategic impact* is concerned with the question of whether the organization is able to capitalize on the benefits obtained from reuse by reaching new markets. To achieve a high *strategic impact*, the company must not only be able to achieve reuse benefits, but also sell them successfully to potential clients.

Software Quality. A frequent claim is that reuse reduces the number of errors in the final product. Previously used components have already been tested. So, if an application is built from such components, the likelihood of failure is lower than in the case of building software from new untested components [24], [36]. The linkage of the strategy characteristics to *software quality* is investigated. Since a reduction of the error rate is a consequence of reuse, even if obtained without a systematic reuse program, *software quality* is the only one of the three items that measures the extend of reuse that is otherwise not tied to the organizational development program.

4.2 Patterns of Reuse Dimensions

Given the nature of the dimensions there is reason to believe that interactions exist. For example, one might only obtain the full benefit of architecture if one also has high project similarity. There are two ways in which such interactions can be estimated. One can introduce interaction terms in a regression model and then estimate them in a straightforward manner. This assumes, though, that the interactions follow a linear pattern. A more flexible approach is to group companies (respondents) together by their co-occurring patterns of reuse dimensions. Just as clustering the practices helped identify higher-level dimensions of practice, clustering the dimensions into metadimensions will help identify patterns of implementation and, perhaps, intent.

This is made operational in the following manner: The dimensional scores noted above are uniquely associated with each organization; each organization can be characterized by a six-dimensional vector representing their scores on *planning and improvement*, *formalized process*, *management support*, *project similarity*, *object technologies*, and *common architecture*. These vectors can then be analyzed using clustering methods [60] and such an analysis identifies

TABLE 4
Cluster Means of Reuse Dimensions

		Reuse Dimensions						Success Measures		
Cluster	n	Planning and improvement	Formalized process	Management support	Project similarity	Object technologies	Common architecture	Reuse Benefit	Strategic Impact	Software Quality
A	10	1.94	1.60	2.17	4.07	3.12	4.10	3.30	2.26	4.10
B	7	2.45	2.23	2.73	2.52	2.16	1.71	3.00	2.58	3.28
C	14	2.70	2.35	3.33	2.61	2.36	3.65	3.11	2.88	3.36
D	14	2.89	3.40	2.40	3.51	3.28	3.39	3.54	3.06	3.33
E	24	3.71	3.60	3.98	3.71	2.16	4.42	4.25	3.48	3.99

clusters of companies that have similar patterns of reuse implementation. In particular, Ward's Method was used to identify clusters of relatively similar factor values.

Clusters A through E have respective sizes of 10, 7, 14, 14, and 24. The next step is to determine why organizations clustered together as they did. Table 4 shows the mean value of each of the six reuse dimensions for each of the five clusters, as well as the reuse program success by cluster.

4.3 Analysis of Reuse Success

Qualitative analysis can be done to try to understand the underlying patterns. Data in Table 4 was examined and, where significant differences were suspected, t-tests were used. This was done in an iterative fashion until we arrived at the following explanation. The *object technologies* dimension was determined to be insignificant in explaining any of the patterns. Although this result may be surprising at first, it is consistent with object technology practice and research [20], [43]. Both indicate that object-oriented methods do not always lead to high reuse. Further, an organization can succeed at reuse even without employing object-orientated methods. We shall stress that this finding does not contradict the notion of reuse benefiting from object-oriented methods. It merely shows that it takes more than just object-orientation to succeed in it. Thus, the success of a reuse attempt depends on the other characteristics presented in this research.

Cluster E describes organizations that handle reuse in the best possible way. Here, reuse receives management support, is embedded in a formalized and repeatable process, and is planned as well as continuously improved. These organizations leverage the benefits of a common architecture by focusing on a domain of similar projects. Thus, it is not surprising that this cluster is characterized by high values in all three success measures. Performing well in all five (remaining) dimensions leads to success in all of the performance areas—this is a fairly strong message in support of a holistic approach to reuse implementation and management.

Cluster A describes development settings in which there is potential for reuse, but not sufficient organizational support. These organizations develop projects with a high similarity and are using a common architecture, suggesting a high reuse potential. Nevertheless, reuse is not formally integrated in the development process and management support is low. Some reuse is accomplished by the developers in an "ad hoc" fashion without any formal support. Thus, cluster A has moderate reuse benefit, relatively poor strategic impact, but strong software quality. This indicates that most reuse benefits cannot be achieved without planning, a formalized process, and management support. But, even with those essential reuse drivers missing, software quality can be achieved based on project similarity and common architecture. In a narrow domain with a common architecture, an organization will invest more effort into the quality of individual components because the number of times individual components are reused is higher than in wider domains. Further, accumulated experience of developers in a narrow domain also may contribute to a better reliability of the components.

Clusters B, C, and D, are fairly similar to one another. Cluster B has the poorest overall reuse success and is characterized by moderate levels of *planning and improvement*, *formalized process*, and *management support*, and low levels of *project similarity* and *common architecture*; in a sense, it is the opposite of Cluster A. These organizations attempt reuse halfheartedly in an environment that has a low potential for reuse. Process factors are somewhat in place, but the projects and domains are not suitable for reuse. It is

TABLE 5
Reuse Archetypes

Reuse Setting	Organizational Dimensions			Development Environment Dimensions	
	Planning & Improvement	Formalized Process	Mgmt. Support	Project Similarity	Common Architecture
Ad-Hoc Reuse with High Reuse Potential	low	low	low	high	high
Uncoordinated Reuse Attempt with Low Reuse Potential	low	low	medium	medium	low
Uncoordinated Reuse Attempt with High Reuse Potential	medium	low	medium	medium	high
Systematic Reuse with Low Management Support	medium	medium	low	high	medium
Systematic Reuse with High Management Support	high	high	high	high	high

not surprising that there is little reuse success in such a setting. Cluster C describes settings that are also attempting reuse halfheartedly. It is similar to B, except that the potential for reuse is much higher, as indicated by the use of a *common architecture*. As a consequence, it does slightly better than Cluster B along all of the success measures. This lets us conclude that a common architecture positively affects a reuse effort. A common architecture in an otherwise only moderately supportive reuse environment is not sufficient to obtain overall reuse benefits. Cluster D is similar to C except that it has higher levels of *formalized process* and *project similarity*, and lower levels of *management support*. Thus, it describes settings that have integrating reuse in the development process and are operating in an environment that is suitable to reuse. However, they lack strong management support. The results are similar to Cluster C, except for a much higher *reuse benefit*. This cluster represents a moderately successful reuse attempt. All characteristics have reasonably high values, and the benefit measures reflect this. The reason for Cluster D's performance being below that of Cluster E is a lower planning and improvement, as well as a lower management support score. In summary:

- Performing well in all reuse dimensions leads to all of the reuse benefits.
- Software quality can be realized by a focus on project similarity and common architecture.
- Performing only moderately well, or poorly, across all of the dimensions only leads to moderate or poor reuse success.
- Focusing on formalized process and project similarity can have good overall performance, but not the best without the other dimensions.

5 Conclusion

We have consolidated an array of existing software reuse success factors into six dimensions based on co-occurrences of reuse practices in an empirical data set. The dimensions describe the characteristics of software reuse settings. To find out which distinct archetypes of reuse strategies exist in the real world, we clustered the dimensions into metadimensions identifying implementation patterns. We found five distinct reuse settings that describe different attempts to implement systematic software reuse with varying success. Table 5 presents the five clusters that form these reuse archetypes. The table classifies the settings using the dimensions established by the study.

Although *Object Technologies* was initially a candidate for a reuse dimension, it has not been used in the final classification scheme. As it was determined to be insignificant in explaining reuse success, we concluded that an organization's reuse success is not dependent on the use of object-oriented techniques. Nevertheless, object technologies may be conducive to reuse, yet the other dimensions that make up the model ultimately determine reuse success. The qualitative analysis of the clusters' reuse success yielded additional insights: While an improvement of software quality can be achieved without an emphasis on the reuse process, an organization will only obtain the full benefit of reuse if a formal reuse program is employed and subject to quality control through formal planning and continuous improvement.

The classification scheme can serve as a framework for future research to differentiate between different reuse strategies, enabling researchers to test more focused theories on reuse strategies employed in practice. While potential reuse adopters can learn which aspects are most crucial to the success of their reuse efforts, particularly to their expectations regarding the benefits of their reuse program, there is a need for further analysis with additional data sets. Specifically, by posing the right questions about

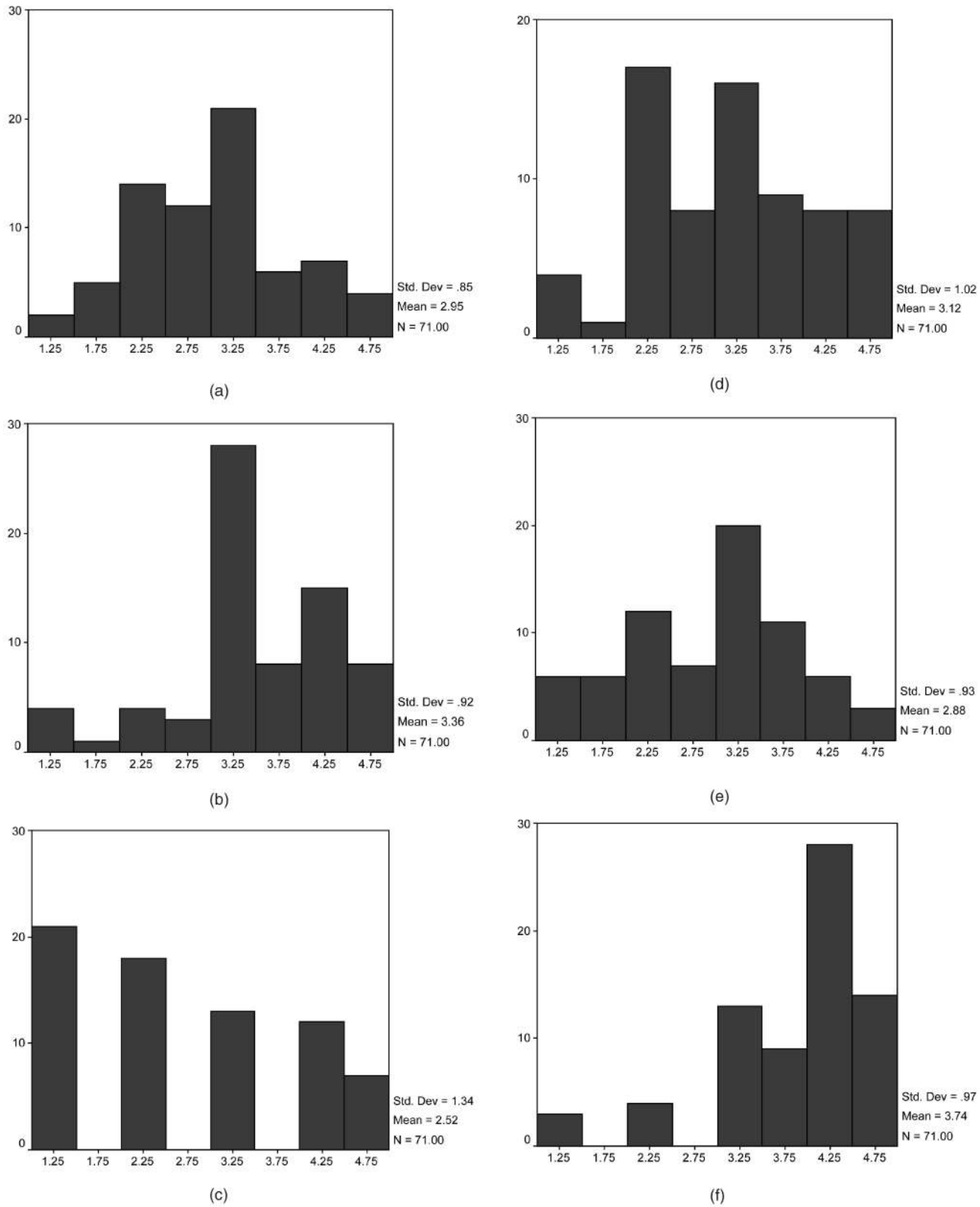

Fig. 1. (a) Planning and improvement. (b) Project similarity. (c) Object technologies. (d) Management support. (e) Formalized process. (f) Common architecture.

different aspects of the now identified reuse dimensions, it may be possible to better measure the actual effort that organizations invest in reuse. Our future research is aimed in these directions.

APPENDIX

FACTOR HISTOGRAMS

The factor histograms are illustrated in Fig. 1.

REFERENCES

[1] U. Apte, C.S. Sankar, M. Thakur, and J.E. Turner, "Reusability-Based Strategy for Development of Information Systems: Implementation Experience of a Bank," *MIS Quarterly*, vol. 14, no. 4, pp. 420-433, 1990.

[2] J.S. Armstrong and T.S. Overton, "Estimating Non-Response Bias in Mail Surveys," *J. Marketing Research*, vol. 14, no. 3, pp. 396-402, 1977.

[3] C. Baldwin and K. Clark, "Managing in the Age of Modularity," *Harvard Business Rev.*, pp. 84-93, Sept.-Oct. 1997.

[4] R.D. Banker and R.J. Kauffman, "Reuse and Productivity in Integrated Computer-Aided Software Engineering: An Empirical Study," *MIS Quarterly*, vol. 15, no. 3, pp. 375-401, 1991.

[5] R.D. Banker, R.J. Kauffmann, C. Wright, and D. Zweig, "Automating Output Size and Reuse Metrics in a Repository-Based Computer-Aided Software Engineering (CASE) Environment," *IEEE Trans. Software Eng.*, vol. 20, no. 3, pp. 169-187, Mar. 1994.

[6] J. Barney, "Firm Resources and Sustained Competitive Advantage," *J. Management*, vol. 17, no. 1, pp. 99-120, 1991.

[7] V.R. Basili, L.C. Briand, and W.L. Melo, "How Reuse Influences Productivity in Object-Oriented Systems," *Comm. ACM*, vol. 39, no. 10, pp. 104-116, 1996.

[8] T.J. Biggerstaff, "An Assessment and Analysis of Software Reuse," *Advances in Computers*, M.C. Yovitis, ed., vol. 34, pp. 1-57, 1992.

[9] T.J. Biggerstaff and C. Richter, "Reusability Framework, Assessment and Directions," *IEEE Software*, vol. 4, no. 2, pp. 41-49, 1987.

[10] S. Brown and K. Eisenhardt, "Product Development: Past Research, Present Findings, and Future Directions," *Academy of Management Rev.*, vol. 20, no. 2, pp. 343-378, 1995.

[11] F.A. Cioch, J.M. Brabbs, and L. Sieh, "The Impact of Software Architecture Reuse on Development Processes and Standards," *J. Systems and Software*, vol. 50, no. 3, pp. 221-236, 2000.

[12] T. Davis, "Adopting a Policy of Reuse," *IEEE Spectrum*, pp. 44-48, June 1994.

[13] W.E. Deming, *Out of the Crisis.* MIT Press, 1986.

[14] R. DeVellis, *Scale Development.* Newbury Park: Sage, 1991.

[15] P. Drucker, *The Frontiers of Management.* Harper & Row, 1987.

[16] S.H. Edwards, "The State of Reuse: Perceptions of the Reuse Community," *Software Eng. Notes*, vol. 24, no. 3, pp. 32-36, 1999.

[17] K.M. Eisenhardt and B.N. Tabrizi, "Accelerating Adaptive Processes: Product Innovation in the Global Computer Industry," *Administrative Science Quarterly*, vol. 40, no. 1, pp. 84-110, 1995.

[18] J. Evans and W. Lindsay, *The Management and Control of Quality.* Cincinnati, Oh.: South-Western College Publishing, 1999.

[19] E. Feitzinger and H. Lee, "Mass-Customization at Hewlett-Packard: The Power of Postponement," *Harvard Business Rev.*, pp. 116-121, Jan.-Feb. 1997.

[20] R. Fichman and C. Kemerer, "Object Technology and Reuse: Lessons from Early Adopers," *Computer*, vol. 30, no. 10, pp. 47-59, Oct. 1997.

[21] W.B. Frakes and C.J. Fox, "Sixteen Questions about Software Reuse," *Comm. ACM*, vol. 38, no. 6, pp. 75-91, 1995.

[22] W.B. Frakes and S. Isoda, "Success Factors of Systematic Reuse," *IEEE Software*, vol. 11, pp. 15-19, Sept. 1994.

[23] W.B. Frakes and C. Terry, "Software Reuse: Metrics and Models," *ACM Computing Surveys*, vol. 28, no. 2, pp. 415-435, 1996.

[24] J.E. Gaffney and T.A. Durek, "Software Reuse–Key to Enhanced Productivity: Some Quantitative Models," *Information and Software Technology*, vol. 31, no. 5, pp. 258-267, 1989.

[25] D. Goldenson and H. Herbsleb, "After the Appraisal: A Systematic Survey of Process Improvement, Its Benefits, and Factor for Success," Technical Report SEI/CMM-95-TR-009, Software Eng. Inst., Carnegie-Mellon Univ., 1995.

[26] A. Griffin, "The Effect of Project and Process Characteristics on Product Development Cycle Time," *J. Marketing Research*, vol. 34, no. 1, pp. 24-35, 1997.

[27] M. Griss and K. Wentzel, "Hybrid Domain Specific Kits for a Flexible Software Factory," *Proc. Ann. ACM Symp. Applied Computing*, pp. 47-52, 1994.

[28] W. Hayes and D. Zubrow, "Moving On Up: Data and Experience Doing CMM-Based Software Process Improvement," Technical Report CMU/SEI-95-TR-008, Software Eng. Inst., Carnegie Mellon Univ., 1995.

[29] A.J. Incorvaia and A.M. Davis, "Case Studies in Software Reuse," *Proc. 14th Ann. Int'l Computer Software & Applications Conf.*, 1990.

[30] S. Isoda, "Experience Report of Software Reuse Projects: Its Structure, Activities, and Statistical Results," *Proc. 14th Int'l Conf. Software Eng.*, pp. 320-326, 1992.

[31] Y. Kim and E.A. Stohr, "Software Reuse: Survey and Research Directions," *J. Management Information Systems*, vol. 14, no. 4, pp. 113-147, 1998.

[32] A. Kleiner and G. Roth, "How to Make Experience Your Company's Best Teacher," *Harvard Business Rev.*, Sept.-Oct. 1997.

[33] C.W. Krueger, "Software Reuse," *ACM Computing Surveys*, vol. 24, no. 2, pp. 131-183, 1992.

[34] L. Latour, E. Johnson, and E. Seer, "A Graphical Retrieval System for Reusable Ada Software Modules," *Proc. Third Int'l Conf. Ada Applications and Environment*, pp. 105-113, 1988.

[35] N.-Y. Lee and C.R. Litecky, "An Empirical Study of Software Reuse with Special Attention to Ada," *IEEE Trans. Software Eng.*, vol. 23, no. 9, pp. 537-549, Sept. 1997.

[36] W.C. Lim, "Effects of Reuse on Quality, Productivity, and Economics," *IEEE Software*, vol. 11, no. 5, pp. 23-30, 1994.

[37] M.H. Meyer and A.P. Lehnerd, *The Power of Product Platforms: Building Value and Cost Leadership.* New York: Free Press, 1997.

[38] H. Mili, F. Mili, and A. Mili, "Reusing Software: Issues and Research Directions," *IEEE Trans. Software Eng.*, vol. 21, no. 6, pp. 528-562, June 1995.

[39] D. Miller and J. Shamsie, "The Resource-Based View of the Firm in Two Environments: The Hollywood Film Studios from 1936 to 1965," *Academy of Management J.*, vol. 39, no. 3, pp. 519-543, 1996.

[40] H. Mintzberg, *The Nature of Managerial Work.* Prentice Hall, 1980.

[41] M. Morisio, C. Tully, and M. Ezran, "Diversity in Reuse Processes," *IEEE Software*, vol. 26, no. 4, pp. 56-63, July/Aug. 2000.

[42] J.C. Nunnally and I.H. Bernstein, *Psychometric Theory*, third ed. McGraw Hill, 1994.

[43] C.M. Pancake, "The Promise and the Cost of Object Technology: A Five-Year Forecast," *Comm. ACM*, vol. 38, no. 10, pp. 33-49, 1995.

[44] M.C. Paulk, B. Curtis, M.B. Chrissis, and C.V. Weber, "Capability Maturity Model for Software, Version 1.1.," Technical Report No. CMU/SEI-93-TR-24, Software Eng. Inst., 1993.

[45] S.L. Pfleeger, "Measuring Reuse: A Cautionary Tale," *IEEE Software*, vol. 22, no. 4, pp. 118-127, July 1996.

[46] M. Porter, *Competitive Advantage.* New York: Free Press, 1985.

[47] J.S. Poulin, "Populating Software Repositories: Incentives and Domain-Specific Software," *J. Systems Software*, vol. 30, no. 3, pp. 187-199, 1995.

[48] J.S. Poulin, *Measuring Software Reuse: Principles, Practices, and Economic Models.* Addison-Wesley, 1997.

[49] J.S. Poulin, J.M. Caruso, and D.R. Hancock, "The Business Case for Software Reuse," *IBM Systems J.*, vol. 32, no. 4, pp. 567-594, 1993.

[50] R. Prieto-Díaz, "Status Report: Software Reusability," *IEEE Software*, vol. 10, no. 3, pp. 61-66, May 1993.

[51] A. Pyster and B. Barnes, "The Software Productivity Consortium Reuse Program," *Proc. IEEE Int'l Conf.: Technologies for the Information Superhighway*, 1988.

[52] M. Ramesh and H.R. Rao, "Software Reuse: Issues and an Example," *Decision Support Systems*, vol. 12, no. 1, pp. 57-77, 1994.

[53] T. Ravichandran, "Software Reusability as Synchronous Innovation: A Test of Four Theoretical Models," *European J. Information Systems*, vol. 8, no. 3, pp. 183-199, 1999.

[54] D.C. Rine and R.M. Sonnemann, "Investments in Reusable Software. A Study of Software Reuse Investment Success Factors," *J. Systems and Software*, vol. 41, no. 1, pp. 17-32, 1998.

[55] D. Robertson and K. Ulrich, "Planning for Product Platforms," *Sloan Management Rev.*, pp. 19-31, Summer 1998.

[56] E. Rogers, *The Diffusion of Innovations.* New York: Free Press, 1995.

[57] H.D. Rombach, "Software Reuse: A Key to the Maintenance Problem," *Information and Software Technology*, vol. 33, no. 1, pp. 86-92, 1991.

[58] M.A. Rothenberger and K.J. Dooley, "A Performance Measure for Software Reuse Projects," *Decision Sciences*, vol. 30, no. 4, pp. 1131-1153, 1999.

[59] M.A. Rothenberger and J.C. Hershauer, "A Software Reuse Measure: Monitoring an Enterprise-Level Model Driven Development Process," *Information & Management*, vol. 35, no. 5, pp. 283-293, 1999.

[60] S. Sharma, *Applied Multivariate Techniques.* John Wiley & Sons, 1995.

[61] G. Sindre, R. Conradi, and E.-A. Karlsson, "The REBOOT Approach to Software Reuse," *J. Systems Software*, vol. 30, no. 3, pp. 201-212, 1995.

[62] A. Sutcliffe, "Domain Analysis for Software Reuse," *J. Systems and Software,* vol. 50, no. 3, pp. 175-199, 2000.
[63] B. Tabrizi and R. Walleigh, "Defining Next-Generation Products: An Inside Look," *Harvard Business Rev.,* pp. 116-124, Nov.-Dec. 1997.
[64] M. Titikonda and S. Rosenthal, "Successful Execution of Product Development Projects: The Effects of Project Management Formality, Autonomy and Resource Flexibility," *Academy of Management Conf. Best Papers Proc.,* 1999.
[65] U.S. GAO, "Best Commercial Practices Can Improve Program Outcomes," Technical Report GAO/T-NSIAD-99-116, 1999.
[66] M. Weber, *The Theory of Social and Economic Organization.* New York: Free Press, 1961.

Marcus A. Rothenberger holds an undergraduate degree in computer science and business from the Darmstadt University of Technology, Germany, and also received the PhD degree in information systems and the MBA degree, both from Arizona State University. He is an assistant professor of management information systems at the University of Wisconsin-Milwaukee. Prior to his graduate studies, he was employed by Deutsche Bank AG in the Information Engineering area. His current research interests include software reusability, performance measurement, ERP adoption, and electronic markets. Dr. Rothenberger has been regularly chairing the minitrack on "Component-Based Software Development and Web Services" at the annual Americas Conference on Information Systems since August 2000. He has published his work in major journals and conferences, such as the *Decision Sciences Journal*, *Communications of the ACM*, the *Journal of Systems and Software*, *Information and Management*, and the International Conference on Information Systems.

Kevin J. Dooley is a professor in the Department of Supply Chain Management in the W.P. Carey School of Business at Arizona State University (ASU). He also has affiliate appointments in the Department of Industrial Engineering at the School of Health Administration and Public Policy and the Hugh Downs School of Human Communication. He teaches courses in the areas of management of technology, quality, project management, and research methods in the undergraduate business, MBA, and PhD programs. His research and teaching interests are in the areas of modeling complex systems, quality management, innovation and new product development, information technology, and organizational change. He is the codirector of the ASU Software Factory, a research and service organization dedicated to providing software development support to ASU research projects. He is also a cofounder and COO of the ASU technology spin-off, Crawdad Technologies, a knowledge management software firm.

Uday R. Kulkarni received the Bachelor's degree in electrical engineering from the Indian Institute of Technology, Bombay, the MBA degree from the Indian Institute of Management, Calcutta, and the PhD degree in management information systems from the University of Wisconsin-Milwaukee. He is an associate professor of information systems at the W.P. Carey School of Business of Arizona State University. Prior to his doctoral studies, he was employed for five years in corporate planning and control areas. His current research interests include the use of relational views for decision-support and application of artificial intelligence techniques to manufacturing problems. He has published articles in the *IEEE Transactions on Knowledge and Data Engineering*, *Decision Sciences Journal*, *Journal of Management Information Systems*, *Decision Support Systems*, and the *European Journal of Operations Research*. He is a member of the IEEE.

Nader Nada received the MS degree in computer science from Moorhead State University in 1993 and the PhD degree in information technology from George Mason University in 1998. In 1991, he received the Computer Science Achievement Award from the American Achievement Academy. From 1995-1997, he was a NASA Fellow at George Mason University. From 1998-2000, he joined the Department of Computer Science at George Mason University as an assistant professor. In 1999, he joined the Department of Software Engineering at the Naval Postgraduate School as a visiting professor. In 2001, he joined the College of Information Systems at Zayed University. In 2003, Dr. Nada joined Sharjah College as an associate professor. His research areas include: improving software development practices, reusable software architectures, digital firm solutions, wireless internet, and mobile applications. He has authored three books, several journal papers, and more than 50 conference publications. In 1999, he was the coprinciple investigator in the research and development of a validated reference model for the software reuse process, funded by a grant from the US National Science Foundation. He has close ties with the software industry on both the international and regional levels and some government organizations including NASA headquarters in the Washington D.C. area through several research activities. He is the founder and chair of the International Symposium on Reusable Architectures and Components for Developing Distributed Information Systems (RACDS) and a member of the IEEE 1517 standard committee on Software Reuse. He chaired the UAE National Committee on Information Technology Education, nominated as a member of the National Committee on the Development of the UAE Primary and Secondary Education and a member of the National Committee on Abu Dhabi Environmental Database.