Characteristics of Software Reuse Strategies: A Taxonomy of Implementations Patterns

Marcus A. Rothenberger[1], Kevin J. Dooley[2], Uday R. Kulkarni[3] and Nader Nada[4]

[1] School of Business Administration; University of Wisconsin – Milwaukee; P.O. Box 742
Milwaukee, WI 53201; USA; +1-414-229-6829; fax +1-414-229-5999; E-mail: rothenb@uwm.edu

[2] *Departments of Management & Industrial Engineering; Arizona State University*; P.O. Box 874006*;*
Tempe, AZ 85287-4006; USA; +1-480-965-4612; fax +1-480-965-8692; E-mail: kevin.dooley@asu.edu

[3] School of Accountancy & Information Management; Arizona State University; P.O. Box 873606
Tempe, AZ 85287-3606; USA; +1-480-965-6191; fax +1-480-965-8392; E-mail: uday.kulkarni@asu.edu

[4] College of Information Systems, Zayed University; P.O. Box 4783
Abu Dhabi; United Arab Emirates; +971-2-4087-857; E-mail: nnada@emirates.net.ae

Abstract - Planned reuse of software artifacts can improve productivity and quality in software development, however, a planned reuse effort requires substantial investments in the process and the software repository. In case of failure of a reuse effort, the initial expenses may not be recovered. The likelihood of success may vary with the reuse strategy employed; hence, potential reuse adopters must be able to understand reuse strategy alternatives and their implications. This research identifies the characteristics of systematic software reuse strategies and evaluates how they contribute to a successful reuse program using survey data collected from seventy-one software development groups. Our results show that these characteristics cluster into five distinct reuse strategies and that each strategy has a different potential for success.

Index Terms - reusability, systematic software reuse, software process improvement, quality, reuse success, reuse classification scheme

1 Introduction

Systematic software reuse is a technique that is being employed to address the need for improvement of software development efficiency and quality [44]. 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]. Traditionally, code and algorithms have been reused in an ad-hoc manner. In contrast, *systematic* software reuse involves building and maintaining a software repository that supports the predictable and timely retrieval of reusable artifacts [19].

Writing reusable software requires additional development effort compared to writing code in a non-reuse setting. Components need to be developed in a generic fashion that allows their use in various contexts. Populating the software repository 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 systematic software reuse involves risk and the choice of a reuse strategy can be crucial to its success. Although Morisio et al. [36] 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 systematic software reuse, and if so, how.

A major reuse effort reported in the literature is the REBOOT (Reuse Based on Object-Oriented Techniques) consortium [55]. 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 [26] confirmed this by arguing that software reuse can only succeed if also non-technical issues are considered. Edwards [13] reports on the results of a survey of reuse experts, which also indicates that additional work is required with respect to the non-technical reuse issues. Our research addresses this need by exploring the practices associated with the reuse process. Lee and Litecky [30] 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. Nevertheless, information about the different practices of reuse and their success cannot be derived from this survey, as it is limited to object-based reuse settings that use one particular programming language. The literature has also discussed broader reuse classification schemes [7, 28, 45] 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 [45] classification scheme characterizes different approaches to reuse along a variety of dimensions: substance, scope, mode, technique, intention, and product. The model considers reuse in a broad sense, including facets, such as automatic code generation and 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 have drawn from existing literature to identify specific practices associated with systematic software reuse,

such as *reuse measurement*, the degree of *commonality of the architecture*, and the level of *management support* [1, 19, 27, 30]. The present research uses these literature-identified reuse practices to develop a classification scheme for systematic software reuse. Since these practices will be identified via empirical data, they can collectively be considered a *taxonomy*.

There is a need to understand how these reuse practices can be aggregated into constructs that describe the differences between systematic reuse strategies. Such insights can help potential reuse adopters in considering how to go adopt systematic software reuse. The study uses survey data from software development groups working with software reuse to extract a set of dimensions along which these strategies differ. The contribution of this research is the identification of such dimensions and their theoretical development. The study also conducts a preliminary investigation on the relevance of the reuse strategy to the success of a reuse program. This is evaluated by examining the linkage between the dimensions and reuse success. Three measures of reuse success are identified and the linkage to the systematic software reuse strategies is established.

The paper is structured as follows. The next section presents the systematic reuse features from the literature that serve as a basis for the survey questions. The data collection is then addressed and the software reuse dimensions are presented. Next, the linkage to the success of reuse is investigated and the limitations of the study are discussed. A conclusion summarizes the results and discusses its implications.

2 Research Approach

The objective of this research is to determine the dimensions that constitute systematic software reuse. In a first step, we synthesize existing literature in order to determine the dominant themes apparent in discussions about reuse. Dimensions can also be identified through empirical studies of actual practice, observing the process of reuse and its constituent tasks. We continue with an empirically grounded approach to enhance the theoretical notion of these dimensions by using survey data in an exploratory fashion. While survey data is most often used for theory testing, with firm theoretical constructs already in hand, in this project survey data was used to enhance the theoretical constructs by allowing empirical data to suggest the final set of reuse dimensions [11]. The software reuse literature was screened for dominant themes to develop the survey items. The items along with the applicable literature references are presented in Table 1. Other demographic information such as company size was also collected.

Insert Table 1 about here

In order to get a sense of the survey items in aggregate, an affinity diagram was composed. The affinity diagram technique [15] collects 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 categories: project similarity [19, 23], reuse planning [19], measurement [43], process improvement [18], formalized process [18],

management support [1], education [18], object technologies [38], and commonality of architecture [19].

Next, a sample population was identified. While comprehensive lists of organizations that develop software are publicly available and easy to access, companies among them that develop software with systematic software reuse are difficult to identify. Software reuse is still not widely adopted. There is no national or international software engineering database or census bureau representing a population of software developers who practice software reuse. It is expected that only a small number of development groups are currently using this paradigm. Hence, the sample has been selected by means of the following approach: First, the survey questions were posted on a web page, since the World Wide Web is easily accessible to software developers. Potential survey participants were identified from three conferences and symposia that extensively covered the topic of software reuse:

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

Further, subscribers to the following distribution lists were asked to participate:

- 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).

Organizations that were interested in reuse were likely to have participated in at least one of the conferences on software reuse that attract professional participants. The use of the e-mail distribution lists ensured that those who could not be reached through conference participation were also included in the sample. These dissemination channels have a worldwide scope, however, their language is English.

The process yielded a total of seventy-seven participating development groups that belong to sixty-seven different organizations. The respondents include project managers, software engineers, developers, software development consultants, and software engineering researchers in profit and non-profit organizations of different sizes and a variety of industries, including telecommunication, financial services, avionics, government, and education. Based on the company and development group information provided by the respondents, there are no duplicate participants within a development group. Almost 80% of the participants were working in organizations with more than 200 employees. For the current study, data from six respondents who omitted more than four questions was excluded, leaving seventy-one responses for the analysis. Among the remaining data sets, very few questions were left unanswered.

The most common methods of dealing with missing response data are listwise or pairwise exclusion. However, these approaches reduce the number of cases used in the analysis. If the number of missing values for each question is under a threshold of 15 percent of the total number of cases [37], a better method is to substitute the missing

values with the mean of the responses to the question. In this dataset, one question had 13 percent, another 10 percent, and the others less than 7 percent missing respondents. The percentage of omitted responses is well below the threshold. Hence, missing values were replaced with the means.

The same persons may have subscribed to multiple e-mail lists. Since we don't know the extent of the overlap between the lists, as well as between the e-mail lists and the conference participants, the limitation of this sampling approach is that the actual size of the sampling population remains unknown. Therefore we cannot assess the non-response rate. Although this could indicate that the results might not be generalizable, the data permit use of a well-accepted method of testing non-response bias to test for significant differences between early and late responses [2]. This method assumes that the answers of late respondents are representative of the answers of non-respondents. In this study, two groups of eighteen responses were selected, representing the first 25 percent and the last 25 percent of the respondents. Their responses to the survey questions were compared by the use of t-tests. There was no statistically significant difference between the two groups. Although this does not completely exclude the possibility of non-response bias, it suggests that it may not affect the results.

3 Dimensions of Systematic Software Reuse Strategies

When practices co-occur (or not) in a systematic pattern of some sort, we may interpret this at least as an association emergent from management action, if not specifically stemming from managerial intention. For example, if the two items pertaining

to *requirements analysis tools* and *domain analysis* are (positively) correlated with one another, it indicates that the presence (or absence) of the use of a requirements analysis tool in practice tends to be observed with the presence (or absence) of domain analysis. The co-occurrence of these practices may be directly intentional, or it may stem from a larger program of practices being implemented systemically and systematically by the group.

Principle component analysis (PCA; sometimes referred to broadly as factor analysis) can be used to identify these patterns of co-occurring practices throughout the entire data set. First, the correlation between each of the items in the survey is correlated. These data are then aggregated into a correlation matrix, where the entry in cell(i,j) is the correlation between items i and j. PCA then performs a rotation of this matrix such that each eigenvector is 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 [11]. 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.

Six different dimensions are identified (components that have an eigenvalue greater than 1.0 are considered significant [11]), and these six dimensions collectively

account for two-thirds of the overall variance, which is 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 [11]. This leads to a fairly unambiguous assignment of items to dimensions, with the exception of items 12 and 15, which were assigned to the dimension in which their loading was highest. Results are shown in Table 2.

Insert Table 2 about here

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 co-variance, 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 [37].

3.1 The Dimensions and Their Relationship to Reuse Success

In this section we shall describe each of the six dimensions resulting from the factor analysis. We will discuss how the theory-based notion of reuse dimensions was modified by the empirical results. Each paragraph will conclude 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.

3.1.1 Planning and Improvement

The empirical analysis indicates that the theory-based themes of 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 themes, 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.

Theory-Based Dimension: 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 Dimension: Measurement

Question 7: Reuse performance measurement.

Theory-Based Dimension: 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 [49]. 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 [8]. A study by the General Accounting Office [59] 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 [25]. 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 [52]. The selection of performance measures depends on individual management goals [40], thus establishing priorities for developers [10].

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 [14] 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.

3.1.2 Formalized Process

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

Theory-Based Dimension: 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 [58]. 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 [22], 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 [42]. 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) [39]; and has been found to have positive benefit on software development project outcomes [21, 24]. This leads to the following proposition.

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

3.1.3 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 for management's support for reuse. The items are shown below.

Theory-Based Dimension: Management Support

Question 13: Senior management support.

Question 14: Reuse champion in management.

Theory-Based Dimension: Education

Question 15: 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 [60], decision making [35], and organizing resources [12]. 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) [5]. By selecting which resources to provide, senior management enacts a strategic-level decision about which resources are valuable and important relative to competitive advantage [41]. 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 [34]. 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 does not only consider 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 [50]. From the above the following proposition is developed.

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

3.1.4 Project Similarity

The empirical results validate the initial theory-based theme 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 Dimension: 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 re-selling 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 [23]. 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 [27]. The proposition reflects the above.

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

3.1.5 Object Technologies

The factor analysis supports the theory-based theme 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 Dimension: 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 [31, 45]. Object-orientation provides a set of techniques that are conducive of the multiple use of software artifacts across projects. Encapsulation forces the developers to clearly define the interfaces of each class to the outside world. The practice to define 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.

3.1.6 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 Dimension: 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 [32, 57]. A common architecture may provide a solution to growing product complexity [3], and may also aid mass-customization and postponement [16].

Organizations that develop a common architecture analyze application domains to identify generic designs that can be used as templates for integrating reusable parts [45]. The reuse of an architecture forces the definition of reuse standards [9]. Standards, such as common interfaces, component size, ways of specifying input/output, and documentation, are among the key factors to reuse success [47]. Often, such standards are operationalized as design templates that can support the identification of suitable reusable components, thus reducing time spent on retrieval [53]. 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 [19]. Based on this the following proposition is developed.

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

3.2 Associations Between the Reuse Dimensions and Reuse Success

The success of a reuse program is a high-level construct that is measured using three variables that were derived from the reuse literature. The 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. The conceptual issues are the dependent variables that measure reuse success in the context of the subsequent analysis. They are *strategic impact*, *reuse benefit*, and *software quality*. 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.

Insert Table 3 about here

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* [6], *quality improvement* [31], and *decreased maintenance effort* [51]. *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 been already 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 [20, 31]. The linkage of the strategy characteristics to *software quality* is investigated.

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 meta-dimensions 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 [54], and such analysis identifies 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, 14, 7, 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.

Insert Table 4 about here

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 [17, 38]. 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 C 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 not surprising that there is little reuse success in such a setting. Cluster D describes settings that are also attempting reuse halfheartedly. It is similar to C, 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 C 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 B is similar to D 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 D 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 B'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.

4 Limitations

A limitation of this study is the lack of control over the sample size. The target population was software development groups that employed systematic software reuse in their development process. Such groups were not easy to identify. Public sources list software development companies, but not the type of methodology used. Survey participants were contacted through several means of distribution ensuring that the sample population approximates the target population. The distribution channels did not allow us to determine the size of the sample, thus making it impossible to assess the non-response rate. While this imposes a limitation on the ability to ensure that a non-response bias does not exist, evidence has been obtained that the results were not affected. A series of t-tests comparing early and late respondents showed that there was no significant difference between the two.

5 Conclusion

The aim of this study was to develop a set of characteristics that allows us to classify Systematic Software Reuse settings. First, the initial set of dimensions was developed based on the software reuse literature. Then, survey data was used to validate and, in some cases, modify the classification. We also clustered the dimensions into meta-dimensions to identify patterns of implementation. We found five distinct reuse settings that describe different attempts to implement systematic software reuse with varying success. The results supported the importance of the strategy characteristics. Table 5 summarizes the systematic software reuse classification scheme that was developed as a result of this study.

Insert Table 5 about here

The classification scheme of systematic software reuse strategies improves our understanding of this important aspect of software development research. Furthermore, the classification scheme can serve as a framework for future research to differentiate between different systematic reuse strategies. This would make it possible to test more focused theories about reuse strategies employed in practice. The results of the meta-factor analysis have provided valuable insights into the importance of the strategic

dimensions of reuse methodologies. 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 datasets. Specifically, by posing the right questions about different aspects of the now identified strategic dimensions, it may be possible to better measure the actual effort that an organization invests in reuse. We also need to be able to better classify and measure benefits of reuse programs. Our future research is aimed in these directions.

References

[1] U. Apte, C.S. Sankar, M. Thakur, J.E. Turner, “Reusability-based strategy for development of information systems: Implementation experience of a bank,” *MIS Quarterly* , vol. 14, no. 4, 1990, pp. 420-433.

[2] J.S. Armstrong and T.S. Overton, “Estimating non-response bias in mail surveys,” *Journal of Marketing Research*, vol. 14, no. 3, 1977, pp. 396-402.

[3] C. Baldwin and K. Clark, “Managing in the age of modularity,” *Harvard Business Review*, Sept-Oct, 1997, pp. 84-93.

[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, 1991, pp. 375-401.

[5] J. Barney, “Firm resources and sustained competitive advantage,“ *Journal of Management.* vol. 17, no. 1, 1991, pp. 99-120.

[6] V.R. Basili, L.C. Briandand and W.L. Melo, “How reuse influences productivity in object-oriented systems,“ *Communications of the ACM*, vol. 39, no. 10, 1996, pp. 104-116.

[7] T. Biggerstaff and C. Richter, “Reusability framework, assessment and directions, “ *IEEE Software*, vol. 4, no. 2, 1987, pp. 41-49.

[8] S. Brown and K. Eisenhardt, “Product development: Past research, present findings, and future directions,“ *Academy of Management Review*, vol. 20, no. 2, 1995, pp. 343-378.

[9] 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, 2000, pp. 221-236.

[10] W.E. Deming, *Out of the Crisis*, MIT Press, Cambridge, MA., 1986.

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

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

[13] S.H. Edwards, “The state of reuse: perceptions of the reuse community, “ *Software Engineering Notes*, vol. 24, no. 3, 1999, pp. 32-36.

[14] K.M. Eisenhardt and B.N. Tabrizi, "Accelerating adaptive processes: Product innovation in the global computer industry, " *Administrative Science Quarterly*, vol. 40, no. 1, 1995, pp. 84-110.

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

[16] E. Feitzinger and H. Lee, "Mass-Customization at Hewlett-Packard: The power of postponement," *Harvard Business Review*, Jan-Feb 1997, pp. 116-121.

[17] R. Fichman and C. Kemerer, "Object technology and reuse: Lessons from early adopers," *IEEE Computer*, vol. 30, October 1997, pp. 47-59.

[18] W.B. Frakes and C.J. Fox, "Sixteen questions about software reuse," *Communications of the ACM*, vol. 38, no. 6, 1995, pp. 75-91.

[19] W.B. Frakes and S. Isoda, "Success factors of systematic reuse," *IEEE Software*, vol. 11, September 1994, pp. 15-19.

[20] J.E. Gaffney and T.A. Durek, "Software reuse - key to enhanced productivity: some quantitative models," *Information and Software Technology*, vol. 31, no. 5, 1989, pp. 258-267.

[21] D. Goldenson and H. Herbsleb, *After the appraisal: a systematic survey of process improvement, its benefits, and factor for success*, SEI/CMM-95-TR-009 Carnegie-Mellon, Software Engineering Institute., 1995.

[22] A. Griffin, "The effect of project and process characteristics on product development cycle time," *J. of Marketing Research*, vol. 34, no. 1, 1997, pp. 24-35.

[23] M. Griss and K. Wentzel, "Hybrid domain specific kits for a flexible software factory," *Proceedings of the ACM-SAC*, Phoenix, AZ, 1994, pp. 47-52.

[24] W. Hayes and D. Zubrow, *Moving on up: data and experience doing CMM-based software process improvement*, CMU/SEI-95-TR-008, Carnegie Mellon Software Engineering Institute, 1995.

[25] S. Isoda, "Experience report of software reuse projects: Its structure, activities, and statistical results," *Proceedings of the 14th International Conference on Software Engineering*, Melbourne, Australia, 1992, pp. 320-326.

[26] Y. Kim and E.A. Stohr, "Software reuse: survey and research directions," *Journal of Management Information Systems*, vol. 14, no. 4, 1998, pp. 113-147.

[27] A. Kleiner and G. Roth, "How to make experience your company's best teacher," *Harvard Business Review*, September-October 1997.

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

[29] L. Latour, E. Johnson and E. Seer, "A graphical retrieval system for reusable Ada software modules," *Proceedings of the Third International Conference on Ada Applications and Environment*, Piscataway, NJ, 1988, pp. 105-113.

[30] N.-Y. Lee and C.R. Litecky. "An empirical study of software reuse with special attention to Ada," *IEEE Transaction on Software Engineering*, vol. 23, no. 9, 1997, pp. 537-549.

[31] W.C. Lim, "Effects of reuse on quality, productivity, and economics," *IEEE Software*, vol. 11, no. 5, 1994, pp. 23-30.

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

[33] H. Mili, F. Mili and A. Mili, "Reusing software: Issues and research directions," *IEEE Transactions on Software Engineering*, vol. 21, no. 6, 1995, pp. 528-562.

[34] D. Miller and J. Shamsie, "The resource-based view of the firm in two environments: The Hollywood film studios from 1936 to 1965," A*cademy of Management Journal*, vol. 39, no. 3, 1996, pp. 519-543.

[35] H. Mintzberg, *The Nature of Managerial Work*, NY. Prentice-Hall, 1980.

[36] M. Morisio, C. Tully and M. Ezran, "Diversity in reuse processes," *IEEE Software*, July/August 2000, pp. 56-63.

[37] J.C. Nunnally and I.H. Bernstein, *Psychometric Theory*, 3rd edition, McGraw Hill: New York, 1994.

[38] C.M. Pancake, "The promise and the cost of object technology: A five-year forecast," *Communications of the ACM*, vol. 38, no. 10, 1995, pp. 33-49.

[39] M.C. Paulk, B. Curtis, M.B. Chrissis and C.V. Weber, *Capability Maturity Model for Software, Version 1.1*., Software Engineering Institute: Technical Report No. CMU/SEI-93-TR-24, 1993.

[40] S.L. Pfleeger "Measuring Reuse: A Cautionary Tale," *IEEE Software*, July 1996, pp. 118-127.

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

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

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

[44] J.S. Poulin, J.M. Caruso and D.R. Hancock, "The business case for software reuse," *IBM Systems Journal*, vol. 32, no. 4, 1993, pp. 567-594.

[45] R. Prieto-Díaz, "Status report: Software reusability," *IEEE Software*, vol. 10, May 1993, pp. 61-66.

[46] A. Pyster and B. Barnes, "The software productivity consortium reuse program," *Proceedings of the COMPCON*, San Francisco, CA, 1988.

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

[48] 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, 1998, pp. 17-32.

[49] D. Robertson and K. Ulrich, "Planning for product platforms," *Sloan Management Review*, Summer 1998, pp. 19-31.

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

[51] H.D. Rombach, "Software reuse: A key to the maintenance problem," *Information and Software Technology*, vol. 33, no. 1, 1991, pp. 86-92.

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

[53] 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, 1999, pp. 283-293.

[54] S. Sharma, *Applied Multivariate Techniques*, John Wiley and Sons, New York, 1995.

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

[56] A. Sutcliffe, "Domain analysis for software reuse," *J. Systems and Software*, vol. 50, no. 3, 2000, pp. 175-199.

[57] B. Tabrizi and R. Walleigh, "Defining next-generation products: An inside look," *Harvard Business Review*, Nov-Dec. 1997, pp. 116-124.

[58] M. Titikonda and S. Rosenthal, "Successful execution of product development projects: The effects of project management formality, autonomy and resource flexibility," *Academy of Management Conference Best Papers Proceedings*, 1999.

[59] U.S. GAO, "*Best commercial practices can improve program outcomes*," GAO/T-NSIAD-99-116, 1999.

[60] M. Weber, *The Theory of Social and Economic Organization*, Free Press, New York, 1961.

Table 1 Characteristic Practices

Reuse Best Practice Category	Ref.	Survey Question(s)
Project Similarity	[19] [23] [46] [47] [48]	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	[18] [19] [28] [29] [33] [48] [56]	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	[19] [27] [43]	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	[18]	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	[18]	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] [27] [30]	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	[18] [27]	15. Our staff has a very competent software reuse education and/or can attend training courses (internal and/or commercial).
Object Technologies	[27] [38]	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	[19] [27]	20. We rely heavily on a common architecture across our product line.

Table 2 Factor Loading

	Factors					
	Planning & Improvement	**Project Similarity**	**Object Technologies**	**Management Support**	**Formalized Process**	**Common Architecture**
Question						
8	**.72531**	-.02217	-.15291	.12648	.06756	.04555
9	**.72474**	.16033	-.10733	.32120	-.18944	.16568
4	**.61777**	-.04877	.03109	.04198	.21329	.50135
5	**.60590**	-.13861	.10773	.00138	.13711	-.08190
7	**.59296**	.01762	-.18424	.33573	.31181	-.22101
6	**.50442**	.09259	-.23422	.25669	.20444	.11184
2	-.11621	**.93252**	.05895	-.06540	.10248	.16797
1	-.01134	**.88345**	.01553	-.14160	.13386	.05675
3	.10339	**.83379**	.06976	.22364	-.07429	-.03682
16	-.10055	.12245	**.84257**	.05317	.01486	-.16493
19	-.03392	.05869	**.75499**	.06515	.01914	-.11246
17	-.17954	-.12122	**.65290**	-.12064	-.00478	.03275
18	.08181	.11129	**.63465**	-.10652	-.06694	.33900
14	.18101	-.08321	.09280	**.86481**	.09127	.13459
13	.23798	.03906	-.08390	**.79639**	.27436	.11964
15	.43799	.05456	-.12109	**.44825**	-.02634	.00675
11	-.01739	.17140	-.03627	.16142	**.85845**	.17215
10	.50060	.11256	-.09750	.23584	**.61110**	-.06145
12	.48061	-.10829	.13698	.02263	**.60677**	.02384
20	.00484	.15979	-.07692	.22576	.09019	**.80716**
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	α = .78	α = .87	α = .72	α = .72	α = .71	N/A

Table 3 Dependent Variables

Dependent Variable	Ref.	Survey Question(s)
Reuse Benefit	[6] [31] [51]	We have seen significant improvement in our primary motives for implementing software reuse.
Strategic Impact	[19]	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	[20] [31]	Our reusable software components have proven through operational use to be highly reliable.

Table 4 Cluster Means of Reuse Dimensions

Cluster	n	Reuse Dimensions						Success Measures		
		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	**14**	2.89	3.40	2.40	3.51	3.28	3.39	3.54	3.06	3.33
C	**7**	2.45	2.23	2.73	2.52	2.16	1.71	3.00	2.58	3.28
D	**14**	2.70	2.35	3.33	2.61	2.36	3.65	3.11	2.88	3.36
E	**24**	3.71	3.60	3.98	3.71	2.16	4.42	4.25	3.48	3.99

Table 5 Classification Scheme for Systematic Software Reuse Settings

Dimensions / Reuse Setting	Planning & Improve-ment	Formalized Process	Mgmt. Support	Project Similarity	Common Architecture
Low Organizational Support, Ad-Hoc Reuse	low	low	low	high	high
Halfhearted Reuse Attempt with Low Reuse Potential	low	low	medium	medium	low
Halfhearted Reuse Attempt with High Reuse Potential	medium	low	medium	medium	high
Systematic Reuse with Little Mgmt. Support	medium	medium	low	high	medium
Highly Successful Systematic Reuse	high	high	high	high	high