Domain Analysis for the Reuse of Software Development Experiences[1]

V. R. Basili*, L. C. Briand, W. M. Thomas***

*** Department of Computer Science**
University of Maryland
College Park, MD, 20742
USA

**** CRIM**
1801 McGill College Avenue
Montreal (Quebec), H3A 2N4

1. Introduction

We need to be able to learn from past experiences so we can improve our software processes and products. The Experience Factory is an organizational structure designed to support and encourage the effective reuse of software experiences [Bas94]. This structure consists of two organizations which separates project development concerns from organizational concerns of experience packaging and learning. The experience factory provides the processes and support for analyzing, packaging and improving the organization's stored experience. The project organization is structured to reuse this stored experience in its development efforts. However, a number of questions arise:

- What past experiences are relevant?
- Can they all be used (reused) on our current project?
- How do we take advantage of what has been learned in other parts of the organization?
- How do we take advantage of experience in the world-at-large?
- Can someone else's best practices be used in our organization with confidence?

This paper describes approaches to help answer these questions. We propose both quantitative and qualitative approaches for effectively reusing software development experiences.

2. A Framework for Comprehensive Software Reuse

The ability to improve is based upon our ability to build representative models of the software in our own organization. All experiences (processes, products, and other forms of knowledge) can be modeled, packaged, and reused. However, an organization's software

[1] This research was in part supported by NASA grant NSG-5123, NSF grant 01-5-24845, and CRIM

experience models cannot necessarily be used by another organization with different characteristics. For example, a particular cost model may work very well for small projects, but not well at all for large projects. Such a model would still be useful to an organization that develops both small and large projects, even though it could not be used on the organization's large projects. To build a model useful for our current project, we must use the experiences drawn from a representative set of projects similar to the characteristics of the current project.

The Quality Improvement Paradigm (QIP)[Bas85,Bas94] is an approach for improving the software process and product that is based upon measurement, packaging, and reuse of recorded experience. As such, the QIP represents an improvement process that builds models or packages of our past experiences and allows us to reuse those models/packages by recognizing when these models are based upon similar contexts, e.g., project and environmental characteristics, relative to our current project. Briefly, the six steps of the QIP are:

1) Characterize the current project and environment
2) Set goals for successful performance and improvement
3) Choose processes and methods appropriate for the project
4) Execute the processes and the measurement plan to provide real-time feedback for corrective action
5) Analyze the data to assess current practices, determine problem areas, and make recommendations for future projects
6) Package the experience in a form suitable for reuse on subsequent projects

The QIP must allow the packaging of context-specific experience. That is, both the retrieval (Steps 2 and 3) and storage (Step 6) steps need to provide the identification of the context in which the experience is useful. Step 1 helps determine this context for the particular project under study. When a project selects models, processes, etc., (step 3), it must ensure that the chosen experience is suitable for the project. Thus the experience packaging (step 6) must include information to allow future projects to determine whether the experience is relevant for use in their context. The problem is that it is not always an easy task to determine which experience is relevant to which project contexts. We would like to reuse the packaged experience if the new project context is "similar" to the projects from which the experience was obtained.

Basili and Rombach present a comprehensive framework for reuse-oriented software development [BR91]. Their model for reuse-oriented software development is shown in Figure 1. It includes a development process model, which is aimed at project development, and a reuse process model, which enables the reuse of the organization’s experiences. The development process will identify a particular need (e.g., a cost model). The reuse process model can then find candidate reusable artifacts (e.g., candidate cost models) from the experience base, selecting (and adapting if necessary) the one most suitable for the particular project. The artifacts may have originated internally (i.e., was developed entirely from past projects in the organization, e.g., the meta-model approach to cost estimation [BB81]), or externally (e.g., the COCOMO cost model [Boe81]). In any event, a evaluation activity of the reuse process is what determines how well the candidate artifact meets the needs specified by the development process.

Our focus in this paper is on the selection and evaluation of reusable artifacts. One approach to do so is to determine “domains” in which a particular experience package may

be reusable. Then, by assessing the extent to which a project is a member of the domain, one can determine whether a given experience package is suitable for that project.

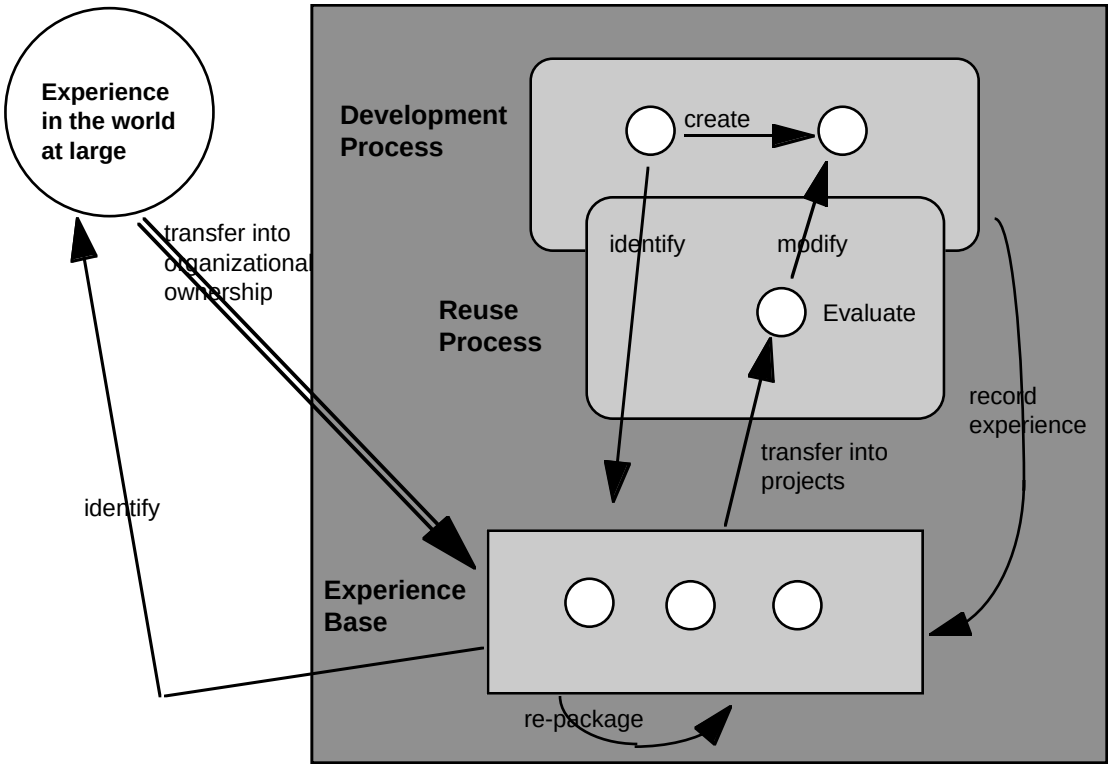

Figure 1: Reuse Oriented Software Development model

3. Experience Domain Analysis

We will use *experience domain analysis* to refer to identifying domains for which reuse of project experiences can be effective, i.e., identifying types of projects for which:

- Similar development or maintenance standards may be applied, e.g., systems for which DoD-Std-2167 is applicable in NASA

- Data and models for cost, schedule, and quality are comparable, e.g., projects for which the productivity can meaningfully be compared within all the branches of NASA

Once domains have been identified, common processes, standards and databases may be shared with confidence by various software organizations, e.g., NASA branches within broader organizational structures such as NASA centers. The problem can be viewed as the

need to determine whether an experience package developed in one context is likely to be effective when reused in a new, different context.

Let us introduce a basic set of definitions to clarify the concepts to be used:

Definition 1: Domain Analysis Goals

> The goal(s) of a domain analysis procedure is (are) to assess the feasibility of reusing or sharing a set of software artifacts within or across organizations, e.g., can we use a cost model developed in another organization, based on a different set of past projects than those of our development organization?

Definition 2: Domains

> With respect to a particular domain analysis goal (i.e., a particular artifact to reuse), *domains* are "types" of software development projects among which certain common artifacts may be reused or shared.

Definition 3: Domain Characteristics and Characterization Functions

> *Domain characteristics* represent characteristics that may determine whether or not one or several artifacts can be reused or shared within or across organizations. The *characterization functions* of domains are mappings of projects, described by characteristics, into domains. For example, in a given organization, large Ada flight simulators may represent a domain with respect to cost modeling and prediction. In this case, the project characteristics involved in the domain characterization function are the project size, the programming language, and the application domain.

It is important to note that for different reuse goals, there are different domains, and as such, different domain characteristics and characterization functions. In this context, the first step of domain analysis is to determine the kind(s) of artifacts one wants to reuse. As an example, someone may want to reuse a cost model or a design inspection procedure developed in a different development environment.

We present in Table 1 a general taxonomy of software artifacts that can conceivably be reused (or shared) within or across organizations. It is important to note that the taxonomy presented here encompasses more than just software products (the usual realm of domain analysis). Also, one could further refine the taxonomy within each specific organization.

Certain kinds of artifacts are more likely to be reusable or sharable than others because they naturally have a broader realm of application. For example, many high-level concepts are universally applicable, e.g., tracking project progress across the development life cycle through data collection helps monitor the schedule and resource consumption of the system being developed. Other kinds of artifacts may have a somewhat more restricted realm of application, e.g., a waterfall process model is applicable as long as the application domain is well known and the solution space is reasonably understood. Also, some kinds of artifact have a very narrow realm of application, e.g., artifacts related to application domain specific programming languages and operating systems (e.g., a real-time UNIX variant for real-time applications).

Data/Models	Standards/Processes	Products
Descriptive models	Requirements	Requirements
Predictive models	Specifications	Specifications
Cost models	Design	Architecture
Schedule models	Coding	Design
Reliability growth models	Testing	Code
Error models	Inspections	Test plans/data
Change models	Change management	
Quality evaluation models		
Lessons learned		
Raw data		

Table 1: Taxonomy of Reusable Software Artifacts

After establishing organizational goals for domain analysis, it is necessary to examine the characteristics of the organization and the projects to help identify appropriate domains. For example, if one wants to reuse a cost model for a new project, the following questions become relevant:

- Are the products developed by the new project comparable to the ones on which the cost model is based ?
- Is the development process similar?
- If not, are the differences taken into account in the model?
- Can the model be imported to this new environment?

The issue is now to determine if the new project belongs to the “same domain” as the projects on which the cost model is built . In order to do so, the characteristics of these projects need to be considered and differences across projects need to be analyzed. Similarly, if one tries to reuse a design inspection process, the following questions are relevant: is staff training sufficient on the new project? Do budget and time constraints allow for the use of such inspections? Is the inspection process thorough enough for the new project reliability requirements?

Table 2 shows a general taxonomy of potentially relevant project characteristics to consider when reusing or sharing software artifacts. The characteristics are grouped according in three broad classes, product, process, and personnel. The table is not intended to be a complete description of all relevant project characteristics, but rather its intent is to provide some guidance as to the characteristics that should be considered. In some environments certain characteristics may not be relevant, and others that are not currently in this table will be. Each organization needs to determine which ones are the most important.

Product	Process	Personnel
Requirements stability	Lifecycle/Process model	Motivation
Concurrent Software	Process conformance	Education
Memory constraints	Project environment	Experience/Training:
Size	Schedule constraints	• Application Domain
User interface complexity	Budget constraints	• Process
Programming language(s)	Productivity	• Platform
Safety/Reliability		• Language
Lifetime requirements		Dev. team organization
Product quality		
Product reliability		

Table 2: Potential Characteristics Affecting Reuse

In the following sections we discuss two techniques to support experience domain analysis, one based on quantitative historical data, and the other based on qualitative expert opinion. Both techniques will provide models that can be used to assess the risk of reusing a particular experience package in a new context.

4. A Quantitative Approach

With sufficient historical data, one can use automated techniques to partition the set of project contexts into domains relative to various levels of reuse effectiveness for a given experience package. The goal of partitioning here is to maximize the internal consistency of each partition with respect to the level of effectiveness. In other words, we want to group projects according to common characteristics that have a visible and significant impact on effectiveness.

Such partition algorithms are based on the multivariate analysis of historical data describing the contexts in which the experience package was applied and the effectiveness of its use. For example, Classification Trees [SP88] or Optimized Set Reduction [BBH93, BBT92] are possible techniques. The measured effectiveness is the dependent variable and the explanatory variables of the domain prediction model are derived from the collection of project characteristics (Table 2) possibly influencing the reuse of the experience package.

As an example, suppose you want to know whether you can use the inspection methodology *I* on project *P*. The necessary steps for answering that question are as follows:

- Determine potentially relevant characteristics (or factors) for reusing *I*, e.g., project size, personnel training, specification formality, etc.
- Determine the measure of effectiveness to be used, e.g., assume the rate of error detection as the measure of effectiveness of *I*. This is used as the dependent variable by the modeling techniques mentioned above.
- Characterize *P* in terms of the relevant characteristics, e.g., the project is large and the team had extensive training.

- Gather data from the experience base characterizing past experience with *I* in terms of the project characteristics and the actual effectiveness. For instance, we may gather data about past inspections that have used methodology *I*.
- Construct a domain prediction model with respect to *I* based on data gathered in the previous step. Then, one can use the model to determine to which domain (i.e., partition) project *P* belongs. The expected effectiveness of *I* on project *P* is computed based on the specific domain effectiveness distribution.

Table 3 shows, for each past project with experience with *I* (A, B, C, D, etc.), the recorded effectiveness of *I* on the project, and a collection of characteristics relevant to the effective reuse of *I* (Size, the amount of training, the formality of the specifications). The last row in the table shows the question that need to be answered for the new project *P*: How effective is *I* likely to be if it is applied to project *P*?

Project	Det. Rate	KSLOC	Training	Formality
A	.60	35	Low	Informal
B	.80	10	High	Formal
C	.50	150	Medium	Informal
D	.75	40	High	Formal
…				
P	***??***	*20-30*	*High*	*Formal*

Table 3: Examples for a Quantitative Approach

From such a table, an example of domain characterization that might be constructed by performing a partition of the set of past projects (more formally: a partition of the space defined by the three project characteristics):

(KSLOC < 50) & (Training=High) => μ(Detection Rate) = 75%

This logical implication indicates that for projects with less than 50 KSLOC and a high level of training, the detection rate is expected to be 75 percent.

When a decision boundary is to be used (e.g., if the level of reuse effectiveness is above 75% of the I's original effectiveness, one reuses I), an alternative form of model is more adequate:

(KSLOC < 50) & (Training=High) => Probability(Detection Rate > 75%) = 0.9

Here the logical implication indicates that for relatively small projects (less than 50 KSLOC) where there is a high level of training, the detection rate with *I* is likely to be greater than 75 percent.

To evaluate the reusability of some experience packages, a quantitative approach to domain analysis is feasible, mathematically tractable, and automatable. Unfortunately, there are a number of practical limitations to such an approach. It is not likely to be effective when:

- There is not sufficient data on past experience
- The effectiveness of reuse is not easily measured
- There is significant uncertainty in the characterization of the new project.

In these cases we may wish to resort to a more heuristic, expert opinion based approach. This is the topic of the next section.

5. A Qualitative Approach

Given the practical limitations to the quantitative approach, a qualitative solution based on expert opinion is needed. As with the quantitative solution, the basic assumption in this approach is that an experience package will be reusable (with similar effectiveness) in a new context if the new context is "similar" to the old context. Rather than defining "similarity" through analysis of historical data, the approach here is to capture and package expert opinion as to what makes an artifact "similar." As we previously noted, what the notion of similarity depends upon the reuse objective. For example, two projects may have widely different cost characteristics (e.g., in the COCOMO classification, one being organic and one embedded) but have very similar error characteristics.

When identifying domains without the support of objective data, one can use several ordinal evaluation scales to determine to what extent a particular characteristic is relevant and usable given a specific reuse goal. These ordinal scales help the analyst introduce some rigor into the way the domain identification is conducted. We propose several evaluation scales that appear to be of importance. The first one determines how relevant is a project characteristic to the use of a given artifact, e.g., is requirements instability a characteristic that can affect the usability of a cost model? A second scale captures the extent to which a characteristic is measurable, e.g., can we measure requirement instability? Other scales can be defined to capture the sensitivity of the metric capturing a domain characteristic, i.e., whether or not a significant variation of the characteristic always translates into a significant variation of the metric, and the accuracy of the information collected about the domain characteristics.

These evaluation scales should be used in this order: (1) determine whether the project characteristic is relevant, (2) assess how well it can be measured, and (3) determine how sensitive and accurate measurement is. If a characteristic is not relevant, then one need not be concerned with whether and how it can be measured. We will define only a relevancy evaluation scale and a measurability evaluation scale here. The two other scales are more sophisticated and beyond the scope of this document.

The relevance scale is intended to indicate the degree to which a characteristic is important with respect to the effective reuse of a particular artifact. The following ordinal scale can be used for this purpose:

1: Not relevant, the characteristic should not affect the use of the artifact of interest in any way, e.g., application domain should not have any effect on the use of code inspections.

2: Relevant only under unusual circumstances, e.g., application domain specific programming language generate the need for application domain specific coding standards whereas, in the general case, the characteristic *application domain* does not usually affect the usability of coding standards.

3: It is clearly a relevant characteristic *but* the artifact of interest can, to some extent, be adjusted so it can be used despite differences in the value. For example, size has an effect on the use of a cost model, i.e., very large projects show lower productivity. Assume that an organization occasionally developing large scale projects wants to reuse the cost model of an organization developing mostly medium-size projects. In this case, the cost model may be calibrated for very large projects. As another example, consider the Cleanroom process [D92]. If not

enough failure data are available during the test phase, the Cleanroom process may be used without its reliability modeling part.

4: It is clearly a relevant characteristic *and* if a project does not have the right characteristic value, the use of the considered artifact is likely to be inefficient. For example, it may not be cost-effective to use specification standards that require formal specifications in a straightforward data processing application domain. Moreover, the artifact is likely to be difficult to tailor.

5: It is clearly a relevant characteristic *and* if a project does not have the right characteristic value, the use of the considered artifact is likely to generate a major failure. For example, if the developed system requires real-time responses to events occurring in its operational environment, requirement analysis and design approaches from a non real-time development environment cannot be used. Moreover, the artifact is likely to be very difficult to tailor.

There are other metrics that are of interest to one interested in the reusability of an artifact. Some of the characteristics may be quite relevant but very difficult to measure. As such, there may be increased risk in the assessment of the reusability of an artifact due to the uncertainty (due to the difficulty in measurement) in the characterization. A measurability scale for the characteristics can be defined as follows:

1: There is no known measure of the characteristic, e.g., development team motivation and morale are hard to measure.

2: There are only *indirect* measures of the characteristic available , e.g., state transition diagram of a user interface can be measured to offer an approximate measure of user interface complexity.

3: There are one or more *direct* measures of the characteristic, e.g., size can be measured on a ratio scale, programming language on a nominal scale.

Two other issues have to be considered when a project characteristic is to be used to differentiate domains. First, we cannot ensure that every significant variation of the characteristic is going to be captured by the measurement, i.e., that the metric is sensitive enough. As a consequence, it may be hard in some cases to tell whether or not two projects are actually in the same domain. Second, a metric may be inherently inaccurate in capturing a characteristic due to its associated data collection process. These two issues should always be considered.

Table 4 shows an example in which we assess the relevance of a subset of the characteristics listed in the taxonomy of domain analysis characteristics, i.e., the product characteristics. Their relevance is considered for the reuse of testing standards, i.e., standards, processes, and procedures defining unit, system, and acceptance test.

Product Characteristic	Relevance Score
Unstable Requirements	3
Concurrent Software	4
Memory Constraints	3
User Interface Complexity	4
Reliability/Safety Requirements	5
Long Lifetime Requirements	3
Product Size	4
Programming Language	1
Intermediate Product Quality	4
Product Reliability	3

Table 4: Product Characteristic Relevancy Scores for the Reuse of Testing Standards

The following paragraphs provide some justification for the relevancy scores assigned for the reuse of testing standards. Detailed justification and explanations about scores for other artifacts will be provided in a subsequent report.

- Unstable or partially undetermined requirements:

  There should be a degree of stability achieved by the start of testing. However, unstable requirements will have a great impact on test planning, and if there are many changes occurring during the test phase, testing will be impacted. Some ways to avoid some of these problems is to have more rigorous inspections, focusing on identifying inconsistencies, and to carefully partition testing activities so that a somewhat stable base is verified, and the impact of the instability is lessened. Also, with the large number of changes late in development, better support for regression test is required. A score of 3 is assigned.

- Concurrent Software:

  In the presence of RT constraints, in addition to verifying functional correctness, it is also needed to verify the necessary performance characteristics of critical threads. These performance requirements must be validated early to allow more options in rectifying problems and to lessen the chance of cost and budget overruns if the requirements are not being met. Also, it would be inefficient for a project that does have such constraints to apply a process that includes early identification and testing of critical threads. A score of 4 is assigned.

• Memory constraints:

As with real-time constraints, problems in meeting memory constraints should be identified early. However, it is typically easier to verify memory use than performance characteristics. It is likely that a standard could be adapted to provide for an earlier verification of the critical memory use. A score of 3 is assigned.

• User interface complexity:

A user interface with a large number of states requires significant verification effort. Certain techniques that are well suited to a smaller number of states (e.g., test every operation in every state) do not scale well to applications with large, complex interfaces. A score of 4 is assigned.

• High Reliability /Safety Requirements:

In safety-critical software, correctness of the implementation is a primary concern. Approaches such as formal verification, fault-tree analyses, and extensive testing are often necessary. A score of 5 is assigned.

• Long Lifetime Requirements:

Testing should be more comprehensive for long-lived software. The goal is not only to ensure current operational suitability, but to allow for operation long after development. With the expectation of a number of changes over the product lifetime, it becomes much more important that the delivered product be thoroughly tested. For example, to ensure a better test coverage, procedures can be put in place to require testing of all paths. A score of 3 is assigned.

• Size:

Certain techniques that are well suited to small applications do not scale well to large applications. More automation and support is likely to be needed. For example, exhaustive path-testing may be useful in smaller applications, but in large applications it is not feasible due to the significant resources that would be required. A score of 4 is assigned.

• Programming Language:

No impact.

• Product quality:

A lesser quality product may be subjected to additional verification procedures so as to ensure a consistent level of quality prior to beginning a certain test activity. This additional verification may not be as cost-effective for products that are known to be of very high quality. For example, applying the same procedures designed for verification of new code to reused software (known to be of high quality) is likely to be less cost-effective. A score of 4 is assigned.

• Reliability:

Testing an unreliable product is a difficult task, as the unreliability may result in a number of changes to correct errors late in development. If one knows that a lower quality product is to be expected (through modeling or comparison with other similar projects), procedures can be used to lessen their impact. For example, more rigorous inspection procedures can be used, targeting the types of defects expected to in the product. Also, additional support for regression testing can be used to help re-integrate changed modules into a particular build. A score of 3 is assigned.

The table can be used in the following manner. For the experience package (e.g., testing standard) of interest, one would examine the table to find which characteristics are of particular importance. Then information about the context of use that characterizes the reusable package in terms of these important characteristics should be obtained. The current project must also be characterized in the same way. These two characterizations can be compared, and a subjective assessment of the risk of reusing the artifact in the context of the new project can be made. The higher the score for a given characteristic on the relevancy evaluation scale, the higher the risk of failure if project sharing or reusing artifacts do not belong to the same domain, i.e., do not show similar values or do not belong to identical categories with respect to a given relevant characteristic. In situations where the risk is high, the reuse/sharing of artifacts will require careful risk management and monitoring[B88]. Sometimes, in order to alleviate the likelihood of failure, the shared/reused artifacts will have to be adapted and modified. Also, if projects appear to belong to the same domain based on an indirect measure (see measurability scale) of the project characteristics, risk can increase due to the resulting uncertainty.

The following example illustrates the approach. The IBM Cleanroom method was considered by NASA/GSFC, code 550, for developing satellite flight dynamics software. In the following paragraph, we give examples of characteristics (and their scores according to the characteristic evaluation scale) that were actually considered before reusing Cleanroom.

- First, it was determined that not enough failure data (Reliability characteristic in Table 2) were produced in this environment in order to build the reliability growth models required by the Cleanroom method. As a consequence, reliability estimates based on operational profiles could not be used to build such models. So Reliability gets a relevancy evaluation metric of 3 and a measurability evaluation metric score of 3.
- There was, despite intensive training, a lack of confidence in the innovative technologies involved in the Cleanroom method, in particular, regarding the elimination of unit test (Personnel Motivation in Table 2: relevancy evaluation score of 3, measurability evaluation score of 1). Therefore, once again, the process was modified: unit test would be allowed if the developer felt it was really necessary and requested it. Interestingly, after gaining experience with the method, it was found that unit test was not being requested, so this change was later removed. Also, there were doubts about the capability of the Cleanroom method to scale up to large projects. As such, the technique was first used on a small scale project (Product Size in Table 2: relevancy evaluation score of 3, measurability evaluation score of 3).
- On the other hand, the use of FORTRAN (versus COBOL in IBM) was not considered as an issue (Programming language in Table 2: relevancy evaluation score of 1, measurability evaluation score of 3).

Once tables such as those shown in Table 4 have been defined for all reuse goals of interest in a given organization, they can be used to help assess whether a software artifact can be reused. For example, suppose that one wants to reuse design standards from other projects on which they appeared to be particularly successful. The relevancy table for design standards may tell us that characteristics such as size and programming language are not very relevant, but that the level of concurrency and real-time in the system are extremely important characteristics to consider.

Suppose that the set of projects where these design standards were assessed as effective can be described as follows: stable requirements, heavy real-time and concurrent software,

no specifically tight memory constraints, and very long lifetime requirements (i.e., long term maintenance). If the project(s) where these standards are to be reused present some differences with respect to some of these important project characteristics, it is likely that the design standards will require some level of tailoring in order to be reusable, if reusable at all. For example, tight memory constraints would generate the need to include in the design standards some strategies to minimize the amount of memory used, e.g., standard procedures to desallocate dynamic memory as soon as possible.

There are a number of weaknesses to this qualitative approach. Perhaps the most important is that it does not adequately express the influence of a factor in a particular context, or, in other words, the interactions between factors. For example, suppose a particular factor, such as tight memory constraints, has an impact on the reusability of a testing methods only in a particular context (e.g., large-scale projects). The table could tell us that tight memory constraints is an important characteristic, but it would not convey the information about the specific context in which it is important. In addition, the table does not quantify the factor's influence on a ratio scale.

6. Conclusions

A wide variety of software experiences are available for reuse within and across most organizations. These experiences may be of local validity (e.g., an error model from NASA/GSFC Code 550), meaningful in a large organization (NASA), or of some value across several development environments (e.g., the COCOMO cost model). However, it is not always clear to what extent the experience package may be reused on a given project. In this paper we described experience domain analysis as an approach to solve this problem. We described two distinct approaches, one quantitative and one qualitative.

The quantitative approach is feasible; however, there are likely to be practical limitations to the approach, primarily due to the difficulty in obtaining sufficient and adequate historical data. The qualitative approach appears more practical; however, it has some drawbacks that may limit its effectiveness. We are working towards a solution that will combine the formality of the quantitative approach with the subjective aspects of qualitative expert opinion. Ideally, we could express rules, derived from expert opinion, which describe the reusability of a package in much the same format as the patterns of the quantitative approach. We are investigating the use of expert systems and fuzzy logic as a means for capturing and representing expert opinion in such a format. Some of the issues with such an approach being addressed include:

- How to acquire expertise?
- How to formalize and package the expert opinion so that it is potentially reusable by other people?
- How to provide a means for dealing with the inherent uncertainty in the expert knowledge?
- How can we check the consistency and completeness of the acquired knowledge?
- How can we combine several expert opinions?

7. References

[B88] B. W. Boehm, Software Risk Management, Prentice–Hall, 1988.

[Bas94] V. Basili et al., "The Experience Factory", Encyclopedia of Software Engineering, Wiley&Sons, Inc., 1994

[BB81] J. W. Bailey and V. R. Basili, "A Meta-model for Software Development Resource Expenditures, *Proceedings of the Fifth International Conference on Software Engineering*, San Diego, 1981.

[Boe81] B. W. Boehm, Software Engineering Economics, Prentice–Hall, 1981.

[BR91] V. R. Basili and H. D. Rombach, "Support for Comprehensive Reuse," *Software Engineering Journal*, 6 (5), September, 1991.

[Bas85] V. Basili, "Quantitative Evaluation of Software Methodology", *Proceedings of the First Pan-Pacific Computer Conference*, Australia, July 1985.

[BR88] V. Basili and H. Rombach, "The TAME Project: Towards Improvement-Oriented Software Environments", *IEEE Trans. Software Eng.*, 14 (6), June, 1988.

[BBH93] L. Briand, V. Basili and C. Hetmanski, "Developing Interpretable Models with Optimized Set Reduction for Identifying High-Risk Software components", *IEEE Trans. Software Eng.*, 19 (11), November, 1993.

[BBT92] L. Briand, V. Basili and W. Thomas, "A Pattern Recognition Approach for Software Engineering Data Analysis", *IEEE Trans. Software Eng.*, 18 (11), November, 1992.

[D92] M. Dyer, The Cleanroom Approach to Quality Software Development, Wiley, 1992.

[SP88] R. Selby and A. Porter, "Learning from Examples: Generation and Evaluation of Decision Trees for Software Resource Analysis", *IEEE Trans. Software Eng.*, 14 (12), December, 1988.