UMIACS–TR–88–92
CS–TR–2158

December, 1988

Towards A Comprehensive Framework for Reuse:† A Reuse–Enabling Software Evolution Environment

V. R. Basili and H.D. Rombach
Institute for Advanced Computer Studies
Department of Computer Science
University of Maryland
College Park, MD 20742

ABSTRACT

Reuse of products, processes and knowledge will be the key to enable the software industry to achieve the dramatic improvement in productivity and quality required to satisfy the anticipated growing demands. Although experience shows that certain kinds of reuse can be successful, general success has been elusive. A software life–cycle technology which allows broad and extensive reuse could provide the means to achieving the desired order–of–magnitude improvements. This paper motivates and outlines the scope of a comprehensive framework for understanding, planning, evaluating and motivating reuse practices and the necessary research activities. As a first step towards such a framework, a reuse–enabling software evolution environment model is introduced which provides a basis for the effective recording of experience, the generalization and tailoring of experience, the formalization of experience, and the (re-)use of experience.

† Research for this study was supported in part by NASA grant nSG–5123, ONR grant N00014–87–K–0307 and Airmics grant DE–AC05–OR21400 to the University of Maryland.

TABLE OF CONTENTS:

1. INTRODUCTION

The existing gap between the demand and our ability to produce high quality software cost–effectively calls for improved software life–cycle technology. A reuse–enabling software life–cycle technology is expected to contribute significantly to higher quality and productivity. Quality can be expected to improve by reusing proven experience in the form of products, processes and knowledge. Productivity can be expected to increase by using existing experience rather than developing it from scratch whenever needed.

Reusing existing experience is the key to progress in any area. Without reuse everything must be re–learned and re–created; progress in an economical fashion is unlikely. During the evolution* of software, we routinely reuse experience in the form of existing products (e.g. generic Ada components, design documents, mathematical subroutines), processes (e.g., design inspections methods, compiler tools), and domain–specific knowledge (e.g., cost models, lessons learned, measurement data). Most reuse occurs implicitly in an ad–hoc fashion rather than as the result of explicit planning and support. While reuse is less institutionalized in software engineering than in other engineering disciplines, there exist some successful cases of reuse, i.e. product reuse. Reuse in software engineering has been successful whenever the reused experience is self–describing, e.g., mathematical subroutines, or the stability of the context in which the experience is reused compensates for the lack of self–description, e.g., reuse of high–level designs across projects with similar characteristics regarding the application domain, the design methods, and the personnel. In software engineering, the potential productivity pay–off from reuse can be quite high since it is inexpensive to store and reproduce software engineering experience compared to other engineering disciplines.

The goal of research in the area of reuse is the achievement of systematic methods for effectively reusing existing experience to maximize quality and cost benefits. Successful reuse depends on the characteristics of the candidate reuse objects, the characteristics of the reuse process

* The term "evolution" is used in this paper to comprise the entire software life–cycle (development and maintenance).

itself, and the technical and managerial environment in which reuse takes place. Interest in reusability has re–emerged during the last couple of years [4, 9, 11, 12, 13, 14, 15, 16, 17, 19, 20, 21], due in part to the stimulus provided by Ada and in part to our increased understanding of the relation between software processes and products.

Our increased understanding tells us that in order to improve quality and productivity via reuse we need a framework which allows (a) the reuse of all kinds of software engineering experience, i.e., products, processes and knowledge, (b) the better understanding of the reuse process itself, and (c) the better understanding of the technical and managerial evolution environment in which reuse is expected to be enabled.

This paper presents a reuse–enabling software evolution environment model, the first step towards a comprehensive framework for understanding, planning, evaluating and motivating reuse practices and the necessary research activities. Section 2 motivates the necessary scope of a comprehensive reuse framework and the important role of a reuse–enabling software evolution environment model within such a framework. Section 3 introduces the reuse–enabling software evolution environment model and discusses its ability to explicitly model the recording of experience, the generalization and tailoring of experience, the formalization of experience, and the (re–) use of experience. The TAME model, a specific instantiation of the reuse–enabling software evolution environment model, is presented in Section 4. This specific instantiation is used to more specifically describe the integration of the recording and (re–)use activities into an improvement oriented software evolution process.

Before we proceed, we define some crucial terms that will be used in this paper so the reader understands what we mean by them in the software context. We have tailored Webster's general definitions of these terms to the specific domain of software evolution. *Improvement* means enhancing a software process or product with respect to quality and productivity. *Learning* is the activity of acquiring experience by instruction (e.g., construction) or study (e.g., analysis). *Reuse* is the activity of repeatedly using existing experience, after reclaiming it, with or without

modification. *Feedback* means returning to the entry point of some process armed with the experience created during prior executions of the process. We use the expression *experience base* to mean a repository containing all kinds of experience. An experience base can be implemented in a variety of ways depending on the type of experience stored. An experience base may consist of one or more of the following: traditional databases containing factual pieces of information, information bases containing structured information, and knowledge bases including mechanisms for deducing new information [5, 24].

2. SCOPE OF A COMPREHENSIVE REUSE FRAMEWORK

Reuse in most environments is implicit and ad-hoc. When it is explicit or planned, it predominantly deals with the reuse of code. In Section 1, we expressed our belief that effective reuse technology needs to be based on (a) the reuse of products, processes and knowledge, (b) a good understanding of the reuse process itself, and (c) a good understanding of the reuse-enabling software evolution environment.

To better justify these beliefs, we will describe and discuss the reuse practice in the Software Engineering Laboratory (SEL) at NASA Goddard Space Flight Center [2, 18]. This is an example where reuse has been quite successful at a variety of levels, albeit predominantly implicit. Ground support software for satellites has been developed for a number of years in FORTRAN. Reused experience exists in the people, methods, and tools as well as in the program library and measurement database.

To explain reuse in this environment we must first explain the management structure. There are two levels of management involved in the technical project management. The second level managers (one from NASA and one from Computer Sciences Corporation, the contractor), have been managing this class of projects for several years. Specific project managers are typically promoted from within the ranks, on either side, from the better developers on prior projects.

This provides a continual learning experience for the management team. Technical review and discussion is informal but commonplace. Lessons learned from experience are used to improve management's ability to monitor and control project developments.

The organizational structure has been relatively constant from project to project. There have been minor variations due to improvements in such things as methods and tools which have evolved from experience or been motivated the literature and verified by experimental data analysis on prior projects.

The basic systems have been relatively constant. This permits reuse of the application knowledge as well as the requirements, and design. For example the requirements documents are quite mixed with regard to the level of specificity. In some places they are quite precise but in other cases the are very incomplete, relying on the experience of the people from prior projects.

Requirements documents have phrases similar to the following: Capability X for new satellite S2 is similar to capability X for satellite S1 except for the following... This implicitly provides reuse of prior requirements documents as well as implicitly allows for reuse of prior design documents and code.

Systems within a class, all have a similar design at the top level and the interfaces among subsystems are relatively well defined and tend to be relatively error free. Design is implicitly reused from system to system as specified by the experienced high level managers.

Reuse at the code level is more explicit. The software development process used is a reuse oriented version of the waterfall model. The coding phase begins by seeding the code library with the appropriately specified elements from the appropriate prior projects. These code components are then examined for their ability to be reused. Some are used as is, others modified minimally, others modified extensively, and yet others are eliminated and judged easier to develop from scratch. This is a reuse approach that has evolved over time and has been quite effective.

A variety of tools have evolved that are quite application specific. These include everything from tools that generate displays needed for testing to application specific system utilities.

Knowledge about these tools has been disseminated by guidance from more senior members of the development team.

The SEL environment is a good example of strong reuse at a variety of levels, in a variety of ways as part of the software development process. There has been a pattern of learning and reusing knowledge, processes and products. The use of the measurement database has helped with project control and schedule as well as quality assessment and productivity [2, 18].

NASA is now considering changing to Ada. Several Ada projects have already been completed. This has involved an obvious loss in the reuse heritage at the code level, as was anticipated. But it has also involved a less obvious and unexpected loss of reuse at the requirements and design level, in the organizational structure, and even in the application knowledge area.

The initial impact of Ada was staggering because of the implicit, rather than explicit, understanding of reuse in the environment. This understanding of reuse needs to be formalized.

Based upon the concept that reuse is more than just reuse of code and that it needs to be explicitly modeled, we need to reconsider how we measure progress in reuse. The measurements currently used in the SEL are based upon lines of code reused from one project to another. Given this view, progress may not be related at all to the lines of code reused. We need to measure the effects of reuse on the resources expended in the entire software life cycle and on the quality of the products produced using an explicit reuse oriented evolution model. In fact, the process should allow us measure for any set of reuse–related goals [3, 4, 8, 10]. Changing our models and our metrics will help us to better understand the effects of the traditional reuse practices and compare them with the effects of an explicit reuse oriented reuse model.

In summary, we believe that a comprehensive reuse framework needs to include (a) a reuse–enabling software evolution environment model, (b) detailed models of reuse and learning, and (c) characterization schemes for reuse and learning based upon these models.

3. A REUSE–ENABLING ENVIRONMENT MODEL

In the past, reuse has been discussed independent of the software evolution environment. We believe reuse can only be an effective mechanism if it is viewed as an integral part, paired with learning, of a reuse–enabling software evolution environment. None of the traditional engineering disciplines has ever introduced the reuse of building blocks as independent of the respective building process. For example, in civil engineering people have not created "reuse libraries" containing building blocks of all shapes and structures, and then tried to use them to build bridges, town houses, high–rises and cottages. Instead, they devised a standard technology for building certain types of buildings (e.g., town houses) through a long process of understanding and learning. This allowed them to define the needs for certain standard building blocks at well–defined stages of their construction process. In the software arena we have not followed this approach.

If we accept the premise that effective reuse requires a good understanding of the environment in which it is expected to take place, then we must model reuse in the context of a reuse–enabling software evolution environment. Such a context will allow us to learn how to reuse better. The ultimate expectation is that such improvement would lead to an ever increasing usage of generator–technology during software evolution. The ability to automate the generation of products from other products reflects the ultimate degree of understanding the underlying construction processes. Automated processes are easy to reuse. For example, in building compiler front–ends, we rarely reuse components of other compilers; instead, we reuse the compiler generators which automate the entire process of building compiler front–ends from formal language specifications.

In Section 3.1 we discuss how learning and reuse implicitly occur in the context of traditional software evolution environments. In Section 3.2, we discuss how learning and reuse can be explicitly modeled in the context of a reuse–enabling software evolution environment.

3.1. Implicit Learning and Reuse

During a workshop on "Requirements for Software Development Environments", held at the University of Maryland in 1985, a view of a software evolution environment was proposed that consisted of an information system and three information producers and consumers: people, methods, and tools [22]. The information system is defined by a software evolution process model describing the information, the communication among people, methods and tools, and the activity sequences for developing and maintaining software.

The traditional software evolution environment model in Figure 1 is a refinement of this earlier model.

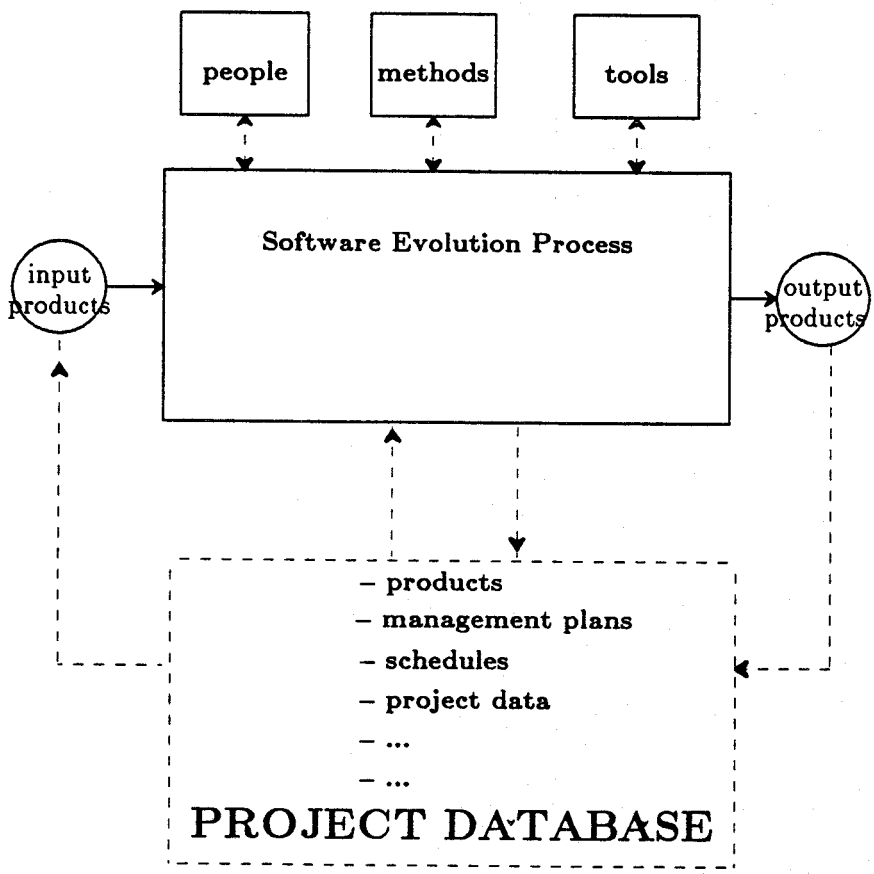

Figure 1: Traditional (non–reuse oriented) Software Evolution Environment Model

The purpose of the software evolution process is to produce output products, e.g., design documents, code, from input products, e.g., requirement documents. People execute this process manually or by utilizing available methods and tools. These methods and tools can be under the control of a project database. All or part of the information produced during this process is stored in a project database, e.g., products, plans such as management plans or schedules, project data.

Typically, support for such a traditional software evolution environment model includes a project database and means for the interaction of people with methods, tools, and the project database during software evolution. The experience of people, as well as some of the methods and tools, is usually not controlled by the project database. As a consequence, this experience is not owned by the organization (via the project database) but rather owned by individual human beings and lost entirely after the project has been completed.

Although the ideas of learning and reuse are not explicitly reflected in the traditional software evolution environment model, they do exist implicitly. The experience of the people involved in the software evolution process and the experience encoded in methods and tools is reused. In many cases, previously developed products are reused as input products. In the same way, products developed during one activity of the evolution process can be reused in subsequent activities of this same process. People learn (gain experience) from performing the activities of the evolution process. Another form of implicit learning occurs whenever products, plans, or project data are stored in the project database.

The basic problem in this traditional environment model is not that learning and reuse can not occur, but that learning and reuse are not explicitly supported and only because of individual efforts or by accident.

3.2. Explicit Modeling of Learning and Reuse

Systematic improvement of software evolution practices requires a reuse–enabling environment model which explicitly models learning, reuse and feedback activities, and integrates them into the software evolution process. Figure 2 depicts such a reuse–enabling environment model.

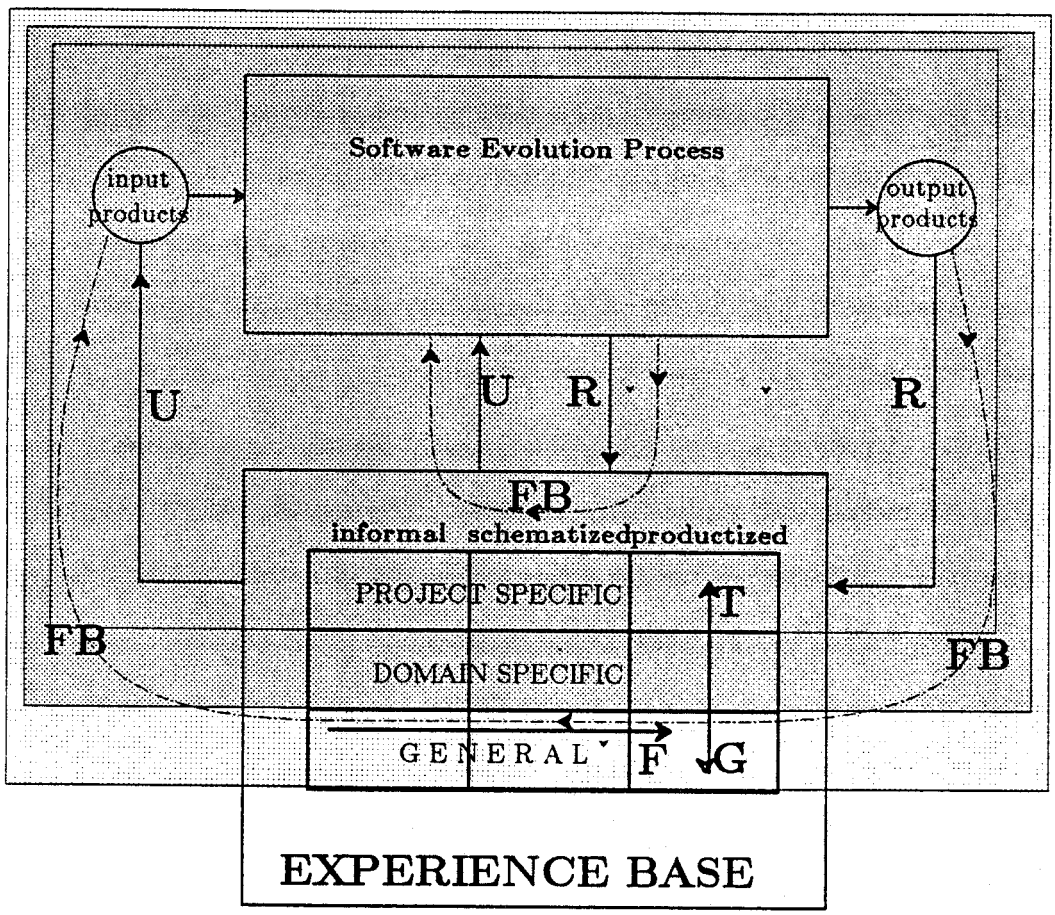

Figure 2: Reuse–Enabling Software Evolution Environment Model

All the potentially reusable experience, including software evolution methods and tools, are under the control of an experience base. Improvement is based on the feedback of existing experience (labeled with "FB" for reuse in Figure 2). Feedback requires learning and reuse. Systematic learning requires support for the recording of experience (labeled with "R" for recording in Figure

2), the off-line* generalizing or tailoring of experience (labeled with "G" and "T" for generalizing and tailoring in Figure 2), and the formalizing of experience (labeled with "F" for formalizing in Figure 2). Off-line generalization is concerned with movement of experience from project-specific to domain-specific and general; off-line tailoring is concerned with movement of experience from general to domain-specific and project-specific. Off-line formalization is concerned with movement of experience from informal to schematized and productized. Systematic reuse requires support for (re-)using existing experience (labeled with "U" for use in Figure 2), and on-line* generalizing or tailoring of candidate experience (not explicitly reflected in Figure 2, because it is assumed to be an integral part of the (re-)use activity).

Although reuse and learning are possible in both the reuse-enabling and the traditional environment models, there are significant differences in the way experience is viewed and how learning and reuse are explicitly integrated and supported. The basic difference between the reuse-enabling model and the traditional model is that learning and reuse become explicitly modeled and are desired characteristics of software evolution.

3.2.1. Recording Experience

The objective of recording experience is to create a repository of well specified and organized experience. This requires a precise description of the experience to be recorded, the design and implementation of a comprehensive experience base, and effective mechanisms for collecting, validating, storing and retrieving experience. We replace the project database of the traditional environment model by an the more comprehensive concept of an experience base which is intended to capture the entire body of experience recorded during the planning and execution of all software projects within an organization. All information flows between the software evolution process and the experience base reflecting the recording of experience are labeled with "R" in Figure 2.

* The attributes "on-line" and "off-line" indicate whether the corresponding activities are performed as part or independent of any particular software evolution project.

Examples of recording experience include such activities as (a) storing of appropriately documented, catalogued and categorized code components from prior systems in a product library, (b) cataloguing of a set of lessons learned in applying a new technology in a knowledge base, or (c) capturing of measurement data related to the cost of developing a system in a measurement database.

In the SEL example of Section 2, code from prior systems is available to the program library of the current project although no code object repository has been developed. Measurement data characterizing a broad number of project aspects such as the project environment, methods and tools used, defects encountered, and resources spent are explicitly stored in the SEL measurement database [2, 8, 18]. Requirements and design documents as well as lessons learned about the technical and managerial implications of various methods and tools are implicitly stored in humans or on paper.

Today it is possible, but not common, to find product libraries. It is even less common to record process–related experience such as process plans or data which characterize the impact of certain methods and tools within an organization. There exist two main reasons why we need to record more process–related experience: (a) it is generally hard to modify existing products efficiently without any knowledge regarding the processes according to which they were created, and (b) the effective reuse of process–related experience such as process plans or data could provide significantly more leverage for improvement than just the reuse of products.

3.2.2. Generalizing & Tailoring Existing Experience Prior to its Potential Reuse

The objective of generalizing existing experience prior to its reuse is to make a candidate reuse object useful in a larger set of potential target applications. The objective of tailoring existing experience prior to its potential reuse is to fine–tune a candidate reuse object to fit a specific task or exhibit special attributes, such as size or performance. These activities require a well–documented cataloged and categorized set of reuse objects, mechanisms that support the

modification process, and an understanding of the potential target applications. Generalization and tailoring are specifically concerned with movement across the boundaries of the "generality" dimension: from general to domain-specific and project-specific and vice versa. Objectives and characteristics are different from project to project, and even more so from environment to environment. We cannot reuse past experience without modifying it to the needs of the current project. The stability of the environment in which reuse takes place, as well as the origination of the experience, determine the amount of tailoring required.

Examples of generalizing and tailoring experience include such activities as (a) developing a generic package from a specific package, (b) instantiating a generic package for a specific type, (c) generalizing lessons learned from a specific design technology for a specific application to any design for that application or any application, (d) or parameterizing a cost model for a specific environment.

In the SEL, requirements and design documents have implicitly evolved to be applicable to all FORTRAN projects in the ground support software domain. Measurement data have been explicitly generalized into domain-specific baselines regarding defects and resource expenditures [2, 8, 18]. Requirements and designs are implicitly tailored towards the needs of a new project based on the manager's experience, and code is explicitly hand-modified to the needs of a new project.

In general, recorded experience is project-specific. In order to reuse this experience in a future project within the same application domain, we have to (a) generalize the recorded project specific experience into domain specific or general experience and (b) then tailor it again to the specific characteristics of the new project. We distinguish between off-line and on-line generalizing and tailoring activities:

- **Off-line generalizing and tailoring** is concerned with increasing the reuse potential of existing process and product-related experience before knowing the precise reuse context (i.e., the project within which the experience is being reused). Off-line generalization and tailoring is

concerned with movement across the boundaries of the specificity dimension within the experience base: from general to domain–specific and then to project–specific, and visa versa. These activities are labeled with "G" and "T" in Figure 2. An example of off–line generalization is the construction of baselines. The idea is to use project–specific measurement data (e.g., fault profiles across development phases) of several projects within some application domain and to create the application–domain specific fault profile baseline. Each new project within the same application domain might reuse this baseline in order to control its development process as far as faults are concerned. An example of off–line tailoring is the adaptation of a general scientific paradigm such as "divide and conquer" to the software engineering domain.

- **On–line tailoring and generalizing** is concerned with tailoring candidate process and product–related experience to the specific needs and characteristics of a project and the chosen software evolution environment. These activities are not explicitly reflected in Figure 2 because they are integral part of the (re–)use activity. An example of on–line tailoring is the adaptation of a design inspection method to better detect the fault types anticipated in the current project [6]. An example of on–line generalization is the inclusion of project specific effort data from a past project into the domain specific effort baseline in order to better plan the required resources for the current project. Obviously, this kind of generalization could have been performed off–line too.

It is important to find a cost–effective balance between off–line and on–line tailoring and generalization. It can be expected that generalization is predominantly performed off–line, tailoring on–line.

A good developer is capable of informally tailoring general and domain specific experience to the specific needs of his or her project. Performing these transformations on existing experience assumes the ability to generalize experience to a broader context than the one studied, or to tailor experience to a specific project. The better this experience is packaged, the better our understanding of the environment. Maintaining a body of experience acquired during a

number of projects is one of the prerequisites for learning and feedback across projects.

A misunderstanding of the importance of tailoring exists in many organizations. These organizations have specific development guidebooks which are of limited value because they "are written for some ideal project" which "has nothing in common with the current project and, therefore, do not apply" [23]. All guidebooks (including standards such as DOD-STD-2167) are general and need to be tailored to each project in order to be effective.

3.2.3. Formalizing Existing Experience Prior to its Potential Reuse

The objective of formalizing existing experience prior to its potential reuse is to increase the reuse potential of a candidate reuse object by encoding it in more precise, better understood ways. This requires models of the various reuse objects, notations for making the models more precise, notations for abstracting reuse object characteristics, mechanisms for validating these models, and mechanisms for interpreting models in the appropriate context. Formalization activities are concerned with movement across the boundaries of the formality dimension within the experience base: from informal to schematized and then to productized. These activities are labeled with "F" in Figure 2.

Examples of formalizing experience include such activities as (a) writing functional specifications for a code module, (b) turning a lessons learned document into a management system that supports decision making, (c) building a cost model empirically based upon the data available, (d) developing evaluation criteria for evaluating the performance of a particular method, or (e) automating methods into tools.

In the SEL, measurement data have been explicitly formalized into cost models [1] and error models enabling the better planning and control of software projects with regard to cost estimation and the effectiveness of fault detection and isolation methods [2, 6, 8, 18]. Lessons learned have been integrated into expert systems aimed at supporting the management decision process [5, 24].

The more we can formalize experience, the better it can be reused. Therefore, we try not only to record experience, but over time to formalize experience from entirely informal (e.g., concepts), to structured or schematized (e.g., methods), or even to completely formal (e.g., tools). The potential for misunderstanding or misinterpretation decreases as experience is described more formally. To the same degree the experience can be modified more easily, or in the case of processes, it may be executed automatically (e.g., tools) rather than manually (e.g., methods).

3.2.4. (Re–) Using Existing Experience

The objective of reusing existing experience is to maximize the effective use of previously recorded experience during the planning and execution of all projects within an organization. This requires a precise characterization of the available candidate reuse objects, a precise characterization of the reuse–enabling environment including the evolution process that is expected to enable reuse, and mechanisms that support the reuse of experience. We must support the (re–)use of existing experience during the specification of reuse needs in order to compare them with descriptions of existing experience, the identification and understanding of candidate, the evaluation of candidate reuse objects, the possible tailoring of the reuse object, the integration of the reuse object into the ongoing software project, and the evaluating of the project's success. All information flows between the experience base and the software evolution process reflecting the (re–)use of experience are labeled with "U" in Figure 2.

Examples of reusing experience include such activities as (a) using code components from the repository, (b) developing a risk management plan based upon the lessons learned from applying a new technology, (c) estimating the cost of a project based on data collected from past projects, or (d) using a development method created for a prior project.

In the SEL, reuse needs are informally specified as part of the requirements document. Matching candidate requirements and design documents are identified by managers who are experienced in this environment. The evaluation of those candidate reuse objects is in part based

on human experience and in part on measurement data. They are tailored based on the application–domain knowledge of the personnel. They are integrated into a very stable evolution process based on human experience. All this reuse is implicit except for the reuse of code, which although explicit, is informal. It could only be successful because it evolved within a very stable environment. The recent change from FORTRAN to Ada has resulted in drastic changes of this environment and as a consequence to the loss in the implicit reuse heritage.

Since the key for improvement of products is always improvement of the process creating those products, we need to put equal emphasis on the reuse of product and process oriented experience. Even today, we have examples of reuse of process experience such as process plans (standards such as DOD–STD–2167, management plans, schedules) or process data (error, effort or reliability data that define baselines regarding software evolution processes within a specific organization). In most of these cases the actual use of this information within a specific project context is not supported; it is up to the respective manager to find the needed information, and to make sense out of it in the context of the current project.

4. TAME: AN INSTANTIATION OF THE REUSE–ENABLING ENVIRONMENT MODEL

The objective of the reuse–enabling software evolution environment model of Section 3.2 is to explicitly model the learning and reuse–related activities of recording experience, generalizing and tailoring experience, formalizing experience, and (re–)using experience so that they can be understood, evaluated, predicted and motivated.

In order to instantiate a specific reuse–enabling environment, we need to choose a model of the software evolution process itself. In general, such an evolution process model needs to be capable of describing the integration of learning and reuse into the software evolution process. In particular, it needs to be capable of modeling when experience is created and recorded into the

experience base as well as when existing experience is used. It needs to provide analysis for the purpose of on–line feedback, evaluating the application of all reuse experience, and off–line feedback for improving the experience base.

The reuse–enabling TAME environment model depicted in Figure 3 is an instantiation of the reuse–enabling software environment model of Section 3.2. based on a very general improvement oriented evolution process model.

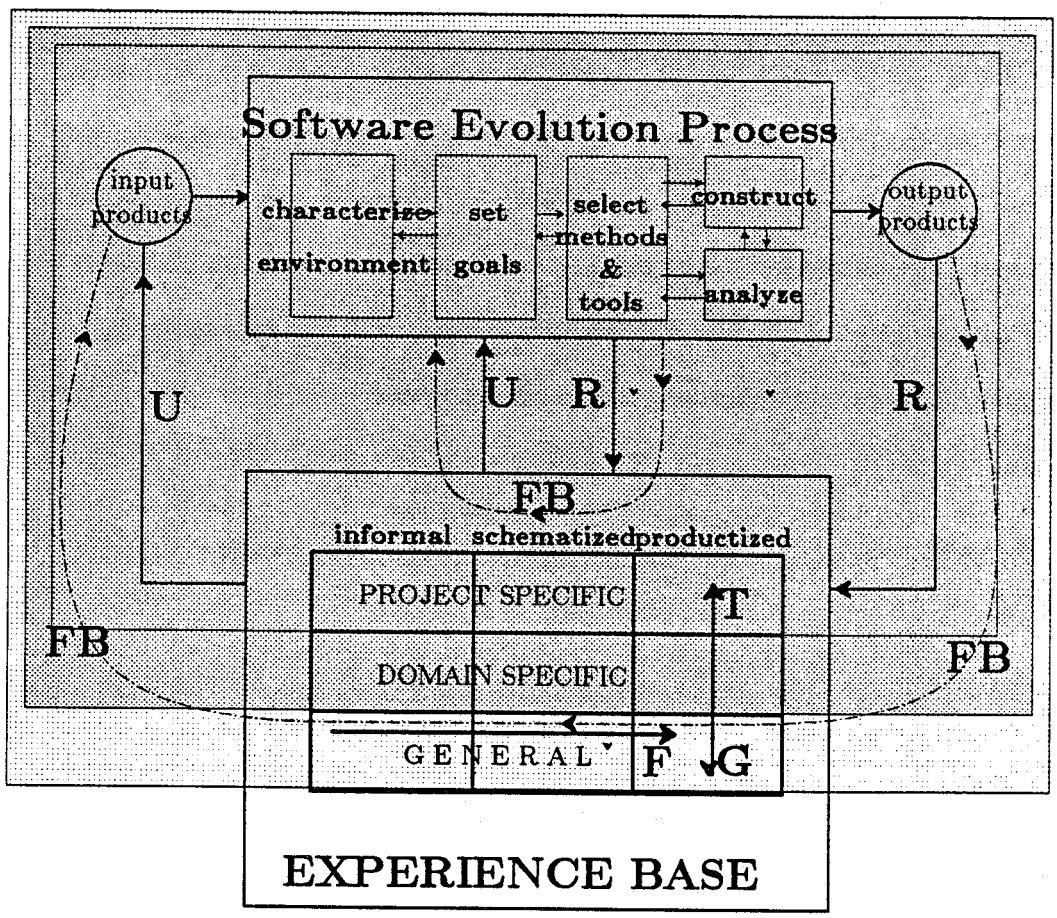

Figure 3: Reuse–Enabling "TAME" Environment Model

Each software project performed according to this improvement oriented evolution process model consists of a planning and an execution stage. The planning stage includes a characteriza-

tion of the current status of the project environment, the setting of project and improvement goals, and the selection of construction and analysis methods and tools that promise to meet the stated goals in the context of the characterized environment. The execution stage includes the construction of output products and the analysis of these construction processes and resulting output products.

The TAME environment model gives us a basis for discussing the integration of the recording and (re–)use activities into the software evolution process. During the environment characterization stage of the improvement oriented process model we (re–)use knowledge about the needs and characteristics of previous projects and record the needs and characteristics of the current project into the experience base. During the goal setting stage we (re–)use existing plans for construction and analysis from similar projects and record the new plans which have been tailored to the needs of the current project into the experience base. During the method and tool selection stage, we (re–)use as many of the constructive and analytic methods and tools which had been used successfully in prior projects of similar type as feasible and record possibly tailored versions of these methods and tools into the experience base. During construction we apply the selected methods and tools, and record the constructed products into the experience base. During analysis we use the selected methods and tools in order to collect and validate data and analyze them, and record the data, analysis results and lessons learned into the experience base.

The TAME environment explicitly supports the capturing of all kinds of experience. The consistent application of the improvement oriented process model across all projects within an organization provides a mechanism for evaluating the recorded experience, helping us to decide what and how to reuse, tailoring and analyzing. TAME supports continuous learning. The explicit and comprehensive modeling of the reuse–enabling evolution environment including the experience base, the evolution process, and the various learning and reuse activities (see Figure 3) allows us to measure and evaluate all relevant aspects of reuse. The measurement methodology used and supported within the TAME environment has been published in earlier papers [7, 8].

5. CONCLUSIONS

In this paper we have motivated and outlined the scope of a comprehensive reuse framework, introduced a reuse–enabling software environment model as a first step towards such a comprehensive reuse framework, and presented a first instantiation of such an environment in the context of the TAME (Tailoring A Measurement Environment) project at the University of Maryland [7, 8].

The reuse–enabling software evolution environment model presented in Section 3 provides a basic environment for supporting the recording of experience, the off–line generalization and tailoring of experience, the off–line formalization of experience, and the (re–) use of existing experience.

Further steps required towards the outlined reuse framework are more specific models of each of these activities that differentiate the components of these activities and serve as a basis for characterization, discussion and analysis. We are currently taking the reuse–enabling software environment model of section 3.2 down one level and developing a model for (re–)using experience. Based on this reuse model we will develop a reuse taxonomy allowing for the characterization of any instance of reuse. The reuse model will provide insight into the other activities of the reuse–enabling environment model only in the way they interact with the (re–)use activity. Corresponding models for each of the other activities need to be developed and integrated into the reuse–enabling software environment model.

The reuse–enabling TAME environment model serves as a basis for better understanding, evaluating and motivating reuse practices and necessary research activities. Performing projects according to the TAME environment model requires powerful automated support for dealing with the large amounts of experience and performing the complicated activities of recording, generalizing and tailoring, formalizing, and (re–)using experience. Indispensable components of such an automated support system are a powerful experience base, and a measurement support system. Many of the reuse approaches in the past have assumed that the developer has sufficient implicit

knowledge of the characteristics of the particular project environment, specific needs for reuse, the candidate reuse objects, etc. It is not trivial to have all this information available. The institutionalized learning of an organization and the proper documentation of that knowledge is definitely one of the keys to effective reuse. This leads to even better specification methods and tools (one of the frequently mentioned keys to effective reuse).

As part of the TAME project at the University of Maryland we have been working on providing appropriate support for building such an experience base, and supporting learning and (re–)use via measurement. We have completed several components towards a first prototype TAME system. These components include the definition of project goals and their refinement into quantifiable questions and metrics, the collection and validation of data, their analysis, and the storage of all kinds of experience. One of the toughest research problems is to use measurement not only for analysis, but also for feedback (learning and reuse) and planning purposes. We need more understanding of how to support feedback and planning. The TAME system is intended to serve as a vehicle for our research towards the effective support of explicit learning and reuse as outlined in this paper.

6. ACKNOWLEDGEMENTS

We thank all our colleagues and graduate students who contributed to this paper by either working on the TAME or any other reuse–related project or reviewing earlier versions of this paper.

7. REFERENCES

[1] J. Bailey, V. R. Basili, A Meta–Model for Software Development Resource Expenditures," in Proc. Fifth International Conference on Software Engineering, San Diego, USA, March 1981, pp. 107–116.

[2] V. R. Basili, "Can We Measure Software Technology: Lessons Learned from Eight Years of Trying," in Proc. Tenth Annual Software Engineering Workshop, NASA Goddard Space Flight Center, Greenbelt, MD, December 1985.

[3] V. R. Basili, "Quantitative Evaluation of Software Methodology," Dept. of Computer Science, University of Maryland, College Park, TR-1519, July 1985 [also in Proc. of the First Pan Pacific Computer Conference, Australia, September 1986].

[4] Victor R. Basili, "Software Maintenance = Reuse-Oriented Software Development," in Proc. Conference on Software Maintenance, Key-Note Address, Phoenix, AZ, October 1988.

[5] V. R. Basili, C. Loggia Ramsey, "ARROWSMITH-P - A Prototype Expert System for Software Engineering Management," IEEE Proceedings of the Expert Systems in Government Symposium, McLean, VA, October 1985, pp. 254-264.

[6] V. R. Basili, H. D. Rombach, "Tailoring the Software Process to Project Goals and Environments," Proc. of the Ninth International Conference on Software Engineering, Monterey, CA, March 30 - April 2, 1987, pp. 345-357.

[7] V. R. Basili, H. D. Rombach, "TAME: Integrating Measurement into Software Environments," Technical Report TR-1764 (or TAME-TR-1-1987), Dept. of Computer Science, University of Maryland, College Park, MD 20742, June 1987.

[8] V. R. Basili, H. D. Rombach "The TAME Project: Towards Improvement-Oriented Software Environments," IEEE Transactions on Software Engineering, vol. SE-14, no. 6, June 1988, pp. 758-773. [is also available as Technical Report (UMIACS-TR-88-8, CS-TR-1983, or TAME-TR-2-1988), Department of Computer Science, University of Maryland, College Park, MD 20742].

[9] V. R. Basili, H. D. Rombach, J. Bailey, and B. G. Joo, "Software Reuse: A Framework," Proc. of the Tenth Minnowbrook Workshop on Software Reuse, Blue Mountain Lake, New York, July 1987.

[10] V. R. Basili, R. W. Selby, D. H. Hutchens, "Experimentation in Software Engineering," IEEE Transactions on Software Engineering, vol.SE-12, no.7, July 1986, pp.733-743.

[11] V. R. Basili and M. Shaw, "Scope of Software Reuse," White paper, working group on 'Scope of Software Reuse', Tenth Minnowbrook Workshop on Software Reuse, Blue Mountain Lake, New York, July 1987 (in preparation).

[12] Ted Biggerstaff, "Reusability Framework, Assessment, and Directions," IEEE Software Magazine, March 1987, pp.41-49.

[13] P. Freeman, "Reusable Software Engineering: Concepts and Research Directions," Proc. of the Workshop on Reusability, September 1983, pp. 63-76.

[14] R. Prieto-Diaz, P. Freeman, "Classifying Software for Reusability," IEEE Software, vol.4, no.1, January 1987, pp. 6-16.

[15] IEEE Software, special issue on 'Reusing Software', vol.4, no.1, January 1987.

[16] IEEE Software, special issue on 'Tools: Making Reuse a Reality', vol.4, no.7, July 1987.

[17] G. A. Jones, R. Prieto-Diaz, "Building and Managing Software Libraries," Proc. Compsac'88, Chicago, October 5-7, 1988, pp. 228-236.

[18] F. E. McGarry, "Recent SEL Studies," in Proc. Tenth Annual Software Engineering Workshop, NASA Goddard Space Flight Center, Greenbelt, MD, Dec. 1985.

[19] Mary Shaw, "Purposes and Varieties of Software Reuse," Proceedings of the Tenth Minnowbrook Workshop on Software Reuse, Blue Mountain Lake, New York, July, 1987.

[20] T. A. Standish, "An Essay on Software Reuse," IEEE Transactions on Software Engineering, vol. SE-10, no. 5, September 1984, pp.494-497.

[21] W. Tracz, "Tutorial on 'Software Reuse: Emerging Technology'," IEEE Catalog Number EHO278-2, 1988.

[22] M. V. Zelkowitz (ed.), "Proceedings of the University of Maryland Workshop on 'Requirements for a Software Engineering Environment', Greenbelt, MD, May 1986," Technical Report TR-1733, Dept. of Computer Science, University of Maryland, College Park, MD 20742, December 1986 [to be published as a book, Ablex Publ., 1988].

[23] M. V. Zelkowitz, R. Yeh, R. Hamlet, J. Gannon, V.R. Basili, "Software engineering practices in the U.S. and Japan," IEEE Computer Magazine, June 1984, pp. 57-66.

[24] J. Valett, B. Decker, J. Buell, "The Software Management Environment," in Proc. Thirteenth Annual Software Engineering Workshop, NASA/Goddard Space Flight Center, Greenbelt, MD, November 30, 1988.