UMIACS–TR–89–57
CS–TR–2263

June, 1989

Software Development: A Paradigm for the Future †

Victor R. Basili
Institute for Advanced Computer Studies
Department of Computer Science
University of Maryland
College Park, MD 20742

ABSTRACT

This paper offers a new paradigm for software development that treats software development as an experimental activity. It provides built–in mechanisms for learning how to develop software better and reusing previous experience in the forms of knowledge, processes and products. It uses models and measures to aid in the tasks of characterization, evaluation and motivation. If proposes an organization scheme for separating the project–specific focus from the organization's learning and reuses focuses of software development. It discusses the implications of this approach for corporations, research and education and presents some research activities currently underway at the University of Maryland that support this approach.

† Research supported in part by NASA grant NSG–5123, AFOSR grant 87–0130, ONR grant N00014–87–K–0307, and ITALSIEL (through the Industrial Associates Program of the Department of Computer Science). Keynote address, COMPSAC '89, Orlando, FL, Sept. 1989.

1. INTRODUCTION

We have been struggling with the problems of software development for many years [31,64]. Organizations have been clamoring for mechanisms to improve the quality and productivity of software. We have evolved from focusing on the project, e.g. schedule and resource allocation concerns, to focusing on the product, e.g. reliability and maintenance concerns, to focusing on the process, e.g. improved methods and process models [27,33,39,56]. We have begun to understand that software development is not an easy task. There is no simple set of rules and methods that work under all circumstances. We need to better understand the application, the environment in which we are developing products, the processes we are using and the product characteristics required.

For example, the application, environment, process and product associated with the development of a toaster and a spacecraft are quite different with respect to hardware engineering. No one would assume that the same educational background and training, the same management and technical environment, the same product characteristics and constraints, and the same processes, methods and technologies would be appropriate for both. They are also quite different with respect to software engineering.

We have not fully accepted the need to understand the differences and learn from our experiences. We have been slow in building models of products and processes and people for software engineering even though we have such models for other engineering disciplines. Measurement and evaluation have only recently become mechanisms for defining, learning, and improving the software process and product [3,34].

We have not even delineated the differences between such terms as technique, method, process and engineering. For the purpose of this paper we define a technique as a basic technology for constructing or assessing software, e.g., reading or testing. We define a method as an organized management approach based upon applying some technique, e.g., design inspections or test plans. We define a process model as an integrated set of methods that covers the life cycle, e.g., an iterative enhancement model using structured designs, design inspections, etc. We define software engineering as the application and tailoring of techniques, methods and processes to the problem, project and organizational characteristics.

There is a basically experimental nature to software development. We can draw analogies from disciplines like experimental physics and the social sciences. As such we need to treat software developments as experiments from which we can learn and improve the way in which we build software.

2. THE IMPROVEMENT PARADIGM

Based upon our experiences in trying to evaluate and improve the quality in several organizations [5,29,53,58], we have concluded that a measurement and analysis program that extends through the entire life cycle is a necessity. Such a program requires an organization to adopt a long term, quality–oriented, organizational life cycle mode, which we call the Improvement Paradigm [4,19]. The paradigm has evolved over time, based upon experiences in applying it to improve various software related issues, e.g., quality and methodology. In its current form, it has four essential aspects:

1 Characterizing the environment. This involves data that characterizes the resource usage, change and defect histories, product dimensions and environmental aspects for prior projects and predictions for the current project. It involves information about what processes, methods and techniques have been successful in the past on projects with these characteristics. It provides a quantitative analysis of the environment and a model of the project in the context of that environment.

2 Planning. There are two integrated activities to planning that are iteratively applied:

   (a) Defining goals for the software process and product operationally relative to the customer, project, and organization. This consists of a top–down analysis of goals that iteratively decomposes high–level goals into detailed subgoals. The iteration terminates when it has produced subgoals that we can measure directly. This approach differs from the usual in that it defines goals relative to a specific project and organization from several perspectives. The customer, the developer, and the development manager all contribute to goal definition. It is, however, the explicit linkage between goals and measurement that distinguishes this approach. This not only defines what good is but provides a focus for what metrics are needed.

   (b) Choosing and tailoring the process model, methods, and tools to satisfy the project goals relative to the characterized environment. Understanding the environment quantitatively allows us to choose the appropriate process model and fine tune the methods and tools needed to be most effective. For example, knowing prior defect histories allows us to choose and fine tune the appropriate constructive methods for preventing those defects during development (e.g. training in the application to prevent errors in the problem statement) and assessment methods that have been historically most effective in detecting those defects (e.g., reading by stepwise abstraction for interface faults).

3 Analysis. We must conduct data analysis during and after the project. The information should be disseminated to the responsible organizations. The operational definitions of process and product goals provide traceability to metrics and back. This permits the measurement to be interpreted in context ensuring a focused, simpler analysis. The goal–driven operational measures provide a framework for the kind of analysis needed. During project

development, analysis can provide feedback to the current project in real time for corrective action.

4 Learning and Feedback. The results of the analysis and interpretation phase can be fed back to the organization to change the way it does business based upon explicitly determined successes and failures. For example, understanding that we are allowing faults of omission to pass through the inspection process and be caught in system test provides explicit information on how we should modify the inspection process. Quantitative histories can improve that process. In this way, hard–won experience is propagated throughout the organization. We can learn how to improve quality and productivity, and how to improve definition and assessment of goals. This step involves the organization of the encoded knowledge into an information repository or expereince base to help improve planning, development, and assessment.

- Characterize the current project environment.
- Set up goals and refine them into quantifiable questions and metrics for successful project performance and improvement over previous project performances.
- Choose the appropriate software project execution model for this project and supporting methods and tools.
- Execute the chosen processes and construct the products, collect the prescribed data, validate it, and and analyze the data to provide feedback in real–time for corrective action on the current project.
- Analyze the data to evaluate the current practices, determine problems, record the findings and make recommendations for improvement for future projects. This is an off–line process which involves the structuring of experience so that it can be reused in the future.
- Proceed to step 1 to start the next project, armed with the recorded, structured experience gained from this and previous projects.

FIGURE 1: THE IMPROVEMENT PARADIGM

The Improvement Paradigm is based upon the assumption that software product needs directly affect the processes used to develop and maintain the product. We must first specify our project and organizational goals and their achievement level. This specification helps determine our processes. In other words, we can't define the processes and then determine how we are going to achieve and evaluate certain project characteristics. We must define the project goals explicitly and quantitatively and use them to drive the process.

As it stands, the improvement paradigm is a generic process whose steps need to be instantiated by various support mechanisms. It requires a mechanism for defining operational goals and transforming them into metrics (step 2a). It

requires a mechanism for evaluating the measurement in the context of the goals (step 3). It requires a mechanism for feedback and learning (step 4). It requires a mechanism for storing experience so that it can be reused on other projects (steps 1,2b). It requires automated support for all of these mechanisms. In the next three sections, we will discuss mechanisms that have been used to support these activities. In the last half of the paper, we will discuss a proposed organizational structure that allows these activities to be managed and evolve.

2.1. The Goal/Question/Metric Paradigm

The Goal/Question/Metric (GQM) paradigm is a mechanism for defining and evaluating a set of operational goals, using measurement on a specific project. It represents a systematic approach for setting the project goals tailored to the specific needs of an organization, defining them in an operational, tractable way by refining them into a set of quantifiable questions that in turn implies a specific set of metrics and data for collection. It involves the development of data collection mechanisms, e.g., forms, automated tools, the collection and validation of data. It includes the analysis of the collected data and computed metrics in the appropriate context of the questions and the original goals.

The GQM paradigm was originally developed for evaluating defects for a set of projects in the NASA/GSFC environment [28]. The application involved a set of case study experiments. It was then expanded to include various types of experimental approaches, including controlled experiments [4,22,25].

The process of setting goals and refining them into quantifiable questions is complex and requires experience. In order to support this process, a set of templates for setting goals, and a set of guidelines for deriving questions and metrics has been developed [19]. These templates and guidelines reflect our experience from having applied the GQM paradigm in a variety of environments.

Goals are defined in terms of purpose, perspective and environment. Different sets of guidelines exist for defining product–related and process–related questions. Product–related questions are formulated for the purpose of defining the product (e.g., physical attributes, cost, changes and defects, user context), defining the quality perspective of interest (e.g., functionality, reliability, user friendliness), and providing feedback from the particular quality perspective. Process–related questions are formulated for the purpose of defining the process (process conformance, domain conformance), defining the quality perspective of interest (e.g., reduction of defects, cost effectiveness of use), and providing feedback from the particular quality perspective.

The GQM provides a mechanism for supporting step 2(a) of the Improvement Paradigm which requires a mechanism for defining operational goals and transforming them into metrics that can be used for characterization, evaluation,

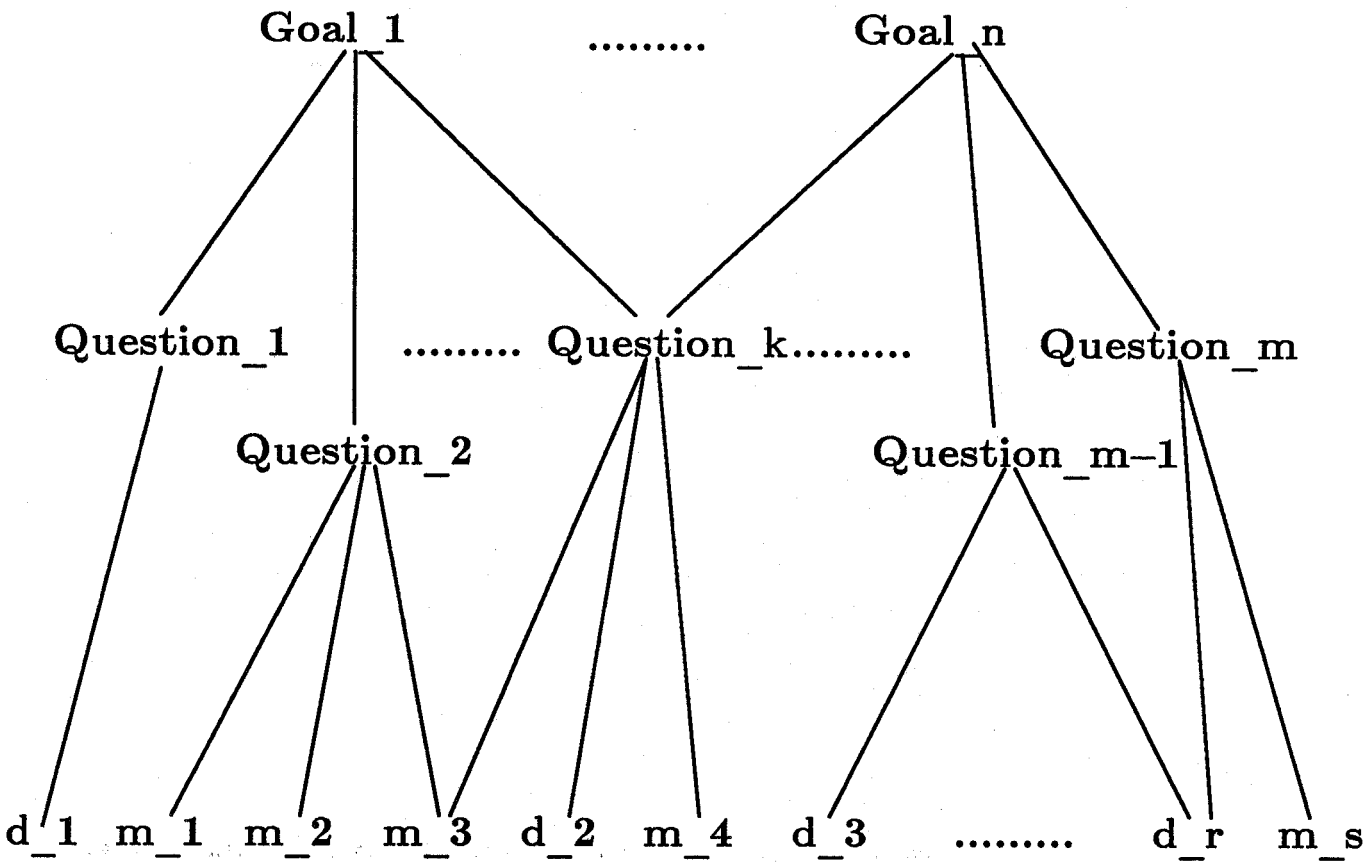

FIGURE 2: THE GOAL/QUESTION/METRIC PARADIGM

prediction and motivation. It supports step 3 by helping to define the experimental context and providing mechanisms for the data collection, validation and analysis activities. It also supports step 4 by providing quantitative feedback on the achievement of goals.

The GQM was originally used to define and evaluate goals for a particular project in a particular environment. In the context of the Improvement Paradigm, the use of the GQM is expanded. Now, we can use it for long range corporate goal setting and evaluation. We can improve our evaluation of a project by analyzing it in the context of several other projects. We can expand our level of feedback and learning by defining the appropriate synthesis procedure for lower–level into higher–level pieces of experience. As part of the IP we can learn more about the definition and application of the GQM in a formal way, just as we would learn about any other experiences.

2.2. The TAME Project

The TAME project [18,19] recognizes the need to characterize, integrate and automate the various activities involved in instantiating the Improvement Paradigm, for use on projects. It delineates the steps performed by the project and

creates the idea of an experience base as the repository for what we have learned during prior developments. It recognizes the need for constructive and analytic activities and supports the tailoring of the software development process.

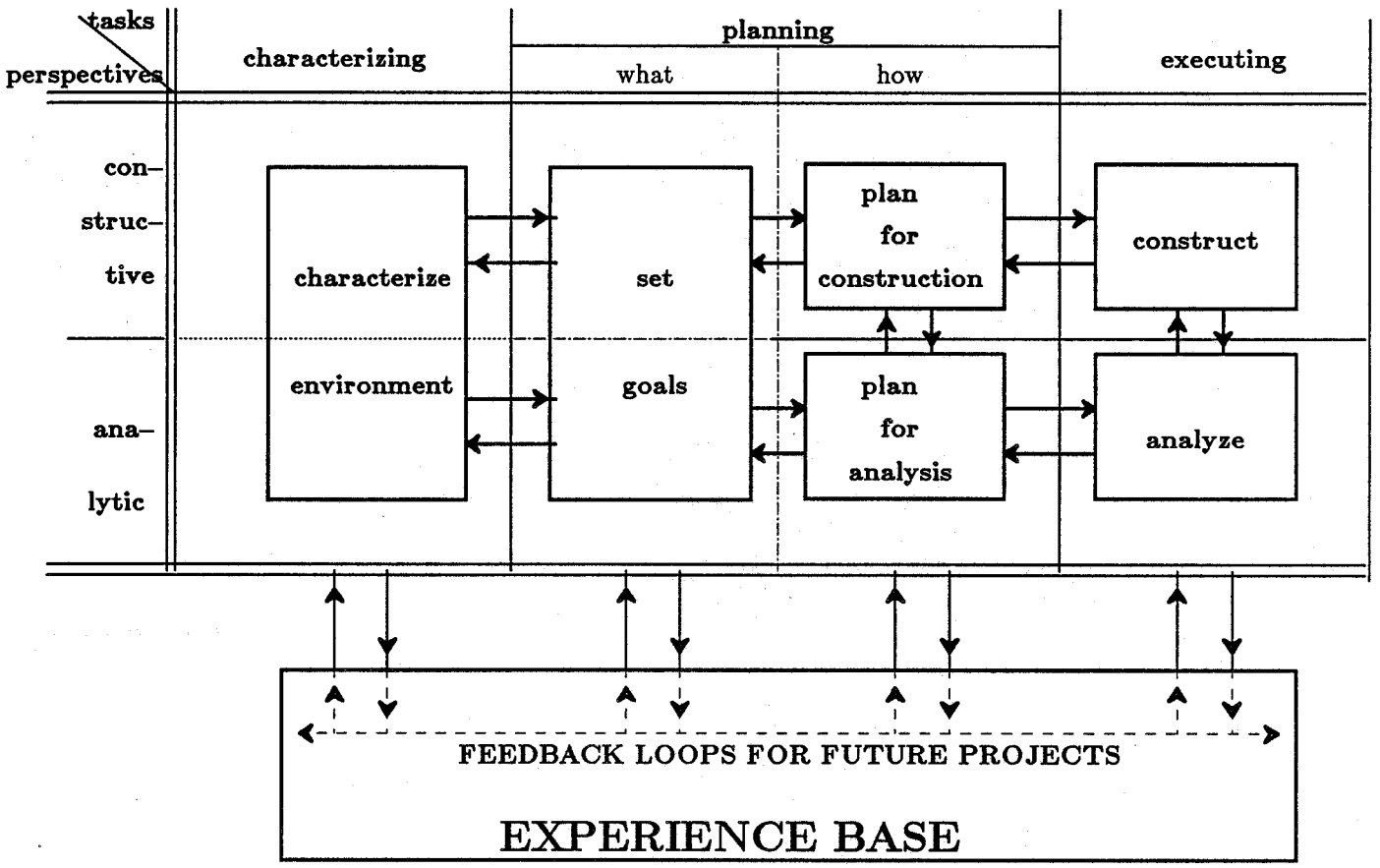

FIGURE 3: THE TAME SYSTEM

The TAME system offers an architecture for a software engineering environment that supports the goal generation, measurement and evaluation activities. It is aimed at providing automated support for managers and engineers to develop project specific goals and specify the appropriate metrics needed for evaluation. It provides automated support for the evaluation and feedback on a particular project in real time as well as help prepare for post mortems.

The Tame project was initiated to understand how to automate as much of the paradigm as possible using whatever current technology is available and to determine where research is needed. It provides a vehicle for defining the concepts in the paradigm more rigorously.

A major goal for the TAME project is to create a corporate experience base which incorporates historical information across all projects with regard to

project, product and process data, packaged in such a way that it can be useful to future projects. This experience base would contain as a minimum the historical data base of collected data and interpreted results, the collection of measured objects, such as project documents, and collection of measurement plans, such as GQM models for various projects. It should also contain combinations and synthesis of this information to support future software development and maintenance.

TAME is an ambitious project. It is assumed it will evolve over time and that we will learn a great deal from formalizing the various aspects of the Improvement Paradigm as well as integrating the various sub–activities. It will result in a series of prototypes, the first of which is to build a simple evaluation environment. Building the various evolving prototypes and applying them in a variety of project environments should help us learn and test out ideas.

Tame provides mechanisms for instantiating the Improvement Paradigm by providing an experience base to allow the storing of experience so that it can be used on other projects (steps 1,2a), further defining the various steps to be performed (steps 1,2,3,4), and automating whatever is possible.

3. A REUSE–ORIENTED SOFTWARE ENGINEERING MODEL

The Improvement Paradigm, as instantiated in the TAME system, assumes that improvement can be achieved by iterating planning, execution of plans, and feedback across projects within an organization. Feedback can be viewed as reusing experience from the ongoing or prior projects to improve the planning or execution of ongoing or future projects. Learning can be viewed as the process of accumulating and packaging experience so it can be reused effectively. Thus, the paradigm explicitly recognizes the need to capture and reuse knowledge, products and processes from prior projects.

On the other hand, it should be noted that reuse can be an effective mechanism only if it is paired with learning and viewed as an integral part of an improvement–oriented software evolution process model. If we accept the fact that a better understanding of a process allows for more effective reuse, "reuse orientation" and "improvement orientation" of a process model are identical attributes. Both are supported by experimentation.

In a traditional software process model, learning and reuse only occur because of individual efforts or by accident. They are not explicitly supported and called out as desired characteristics of the development process. As a consequence, this experience is not owned by the organization (via the project database) but rather owned by individual human beings and lost after the project has been completed. A reuse–oriented process model must view reuse, learning and feedback as integral components, and place all experience, including software evolution methods and tools, under the control of an experience base [20].

Since improvement requires the feedback of available experience and feedback is based on learning and reuse activities, a requirement for such a process model is that it support systematic learning and reuse. Systematic learning requires support for the off–line recording, generalizing or tailoring, and formalizing of experience. Systematic reuse requires support for (re–)using existing experience. Off–line activities are performed independent of any particular project in order to improve the reuse potential of existing experience in the experience base.

Project goals are typically directed towards the development of a specific system. Thus off–line activities must have their own organizational structure. They cannot be part of the normal development organization because they require a different focus, a different set of processes, and an independent cost base.

For example, the objective of the recording process is to create a repository of well specified and organized experience. It requires effective mechanisms for collecting, validating, storing and retrieving experience. This should not be part of the project focus. The project can contribute by making its experience available to this independent organization, but cannot itself oversee the recording. It might not even be clear to the project what is worth recording.

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. Clearly a project cannot afford to generalize or tailor experience for another project within its budget constraints. Even worse, it may not have the perspective to do so since objectives and characteristics are different from project to project, and even more so from environment to environment. Generalizing and tailoring require a broader perspective of the organization and the products it develops.

The objective of formalizing existing experience prior to its potential reuse is to encode it in more precise, better understood ways. Off–line tailored or generalized experience needs to be formalized to increase its reuse potential and satisfy general reuse needs within an organization. The more we can formalize experience, the better it can be reused.

Formalization activities include the movement from informal knowledge (e.g., concepts), to structured or schematized knowledge (e.g., methods), or even to completely formal knowledge or automation (e.g., tools). It 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. Clearly the project has neither the budget nor the need to formalize its own experience.

Reuse requires a precise specification of the reuse context including the evolution process that is expected to enable reuse, and the characteristics of the available candidate reuse objects. The objective of a reuse–oriented software evolution process model is to support the use of previously accumulated experience during such reuse activities as: (a) specifying reuse needs in a way that allows matching them with descriptions of available experience, (b) finding and understanding appropriate reuse candidates, (c) evaluating reuse candidates in order to pick the most promising candidate, (d) actually tailoring the reuse candidate if necessary, (e) integrating the reuse candidate into the ongoing software project, and (f) evaluating the software project.

A reuse–oriented software evolution environment is an integral part of the improvement paradigm. The mechanisms supplied by the TAME system to support that paradigm are consistent with the mechanisms needed to support the reuse environment model with its experience base. It provides a mechanism for evaluating the recorded experience, helping us to decide what and how to reuse, tailor and analyze. It captures experience in the form of data from which models can be built to formalize experience. It supports continuous learning.

It is clear that an experience base is a key component of the reuse and improvement paradigms. A project needs help in accessing the reusable experience. If the experience is available (recorded), appropriate (tailored or generalized), and well–packaged (formalized), it can be used by a project. But an experience base is more than a physical entity. It is an organization that must support all the off–line activities that support its creation and use.

4. DIVIDING UP THE RESPONSIBILITIES AND ACTIVITIES

Based upon the prior discussion, the implementation of the Improvement Paradigm would best be served by two separate and distinct organizational structures. One organization is project–oriented. Its goal is to deliver the systems required by the customer. We will call this the Project Organization. The other organization, which we will call the Experience Factory, will have the role of monitoring and analyzing project developments, developing and packaging experience for reuse in the form of knowledge, processes, tools and products, and supplying it to the Project Organization upon request. The Experience Factory represents the experience base discussed above and the various activities associated with building and modifying it, controlling its access, and interfacing to the Project Organization.

Each project in a Project Organization can choose its process model based upon the characteristics of the project, taking advantage of prior experience with the various process models from the experience base in the Experience Factory. It can access information about prior system requirements and solutions, effective methods and tools and even available system components. Based upon access to this prior experience, the project can choose and tailor the best possible process, methods and tools. It can reuse prior products tailored to its needs.

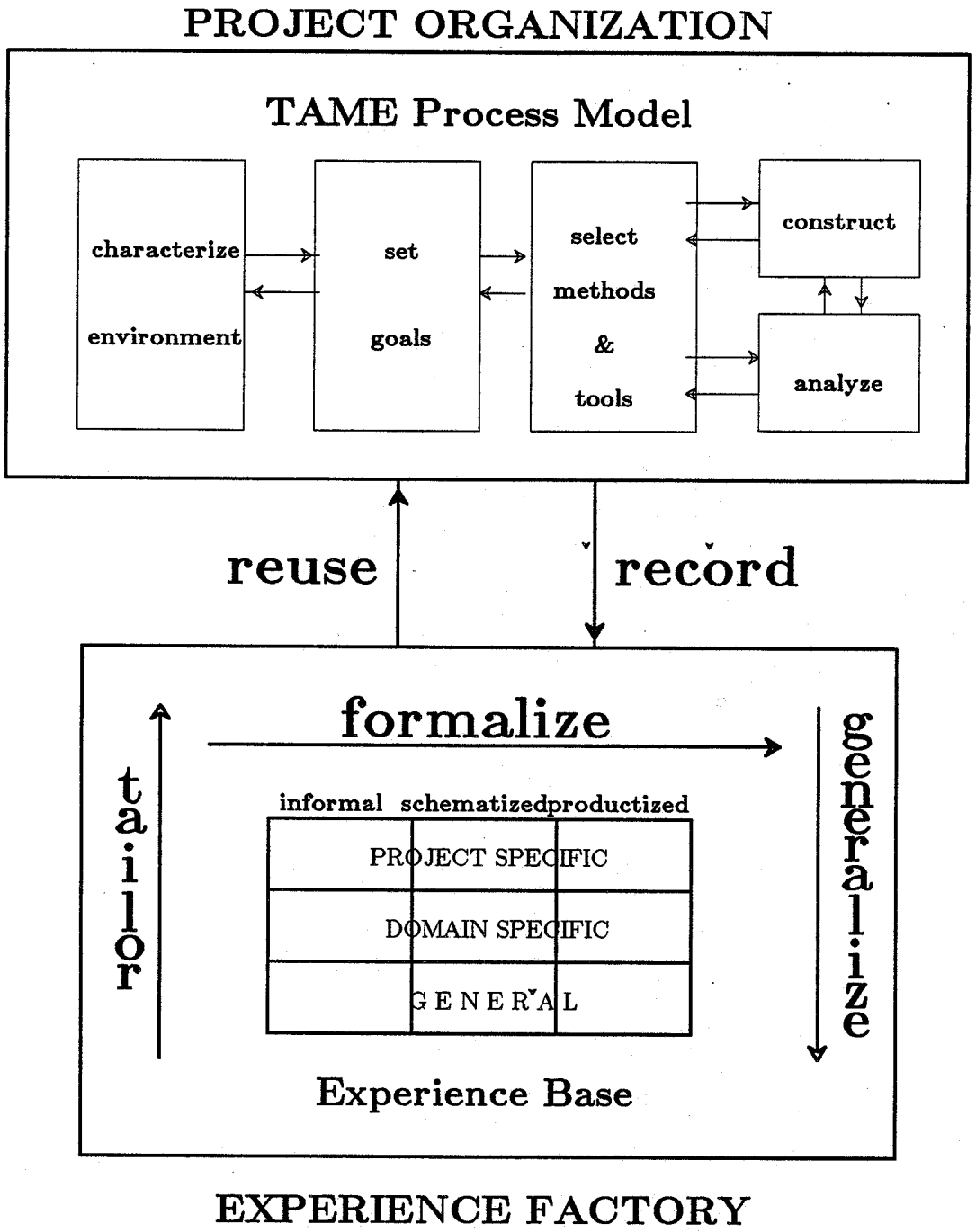

FIGURE 4: IMPROVEMENT AND REUSE ORIENTED–SOFTWARE ENGINEERING MODEL

The Experience Factory analyzes the project development for all systems developed by the corporation. Based upon this it recognizes commonality among projects, generalizes knowledge and packages it for use across all projects. It creates a repository of reusable information. For example, it can develop resource models, defect models, and risk management models and tailor them for the

particular projects. It can develop processes, methods, techniques and tools and tailor them based upon the characteristics of the particular project. This can be accomplished based upon the Factory's analysis of the success and failure of the various activities across many projects. It can generate system components, at various levels of the architectural hierarchy based upon its recognition of commonality.

4.1. Some Specific Activities in the Project Organization

Let us consider the activities of the Project Organization with regard to the development of a system and how it might use the Experience Factory while applying the improvement paradigm.

At the start of a project, project management functions consist of activities such as resource and schedule planning, organizing, and staffing. These are covered by the characterizing and planning functions in the Improvement Paradigm.

During the characterizing phase, based upon its needs and characteristics, the project can access the experience base for the information about similar previous projects. This provides the project manager with a context for planning that includes resource estimation and allocation information, personnel experience, software and hardware available for reuse, environmental characteristics of concern and sets of baselines for resources, schedules, defects, etc. The project can store information on its own characteristics back into the experience base for analysis.

During the planning phase, the project can analyze prior goals and use them as defined or tailor them (or have them tailored by the Component Factory) for its needs. It can access the collection of construtive and analytic methods and tools, that have been effective and choose the appropriate ones that will help satisfy its goals. The goals and methods are influenced by the knowledge gained from the characterization phase, specifically with regard to elements of prior systems that can be reused. These elements include data, such as baselines, process models that have been successful, including methods and techniques that have been tailored and tools that support those methods, and components of prior projects such as requirements, design or code that can be adapted for the current project. The goals define the kinds of data that need to be collected as well as the mechanisms needed for collection. This provides the manager with information about what feedback will be provided for the project during development. The goals and process model, as tailored for the project, are stored in the experience base for monitoring the current project and expanding the experience base for future projects.

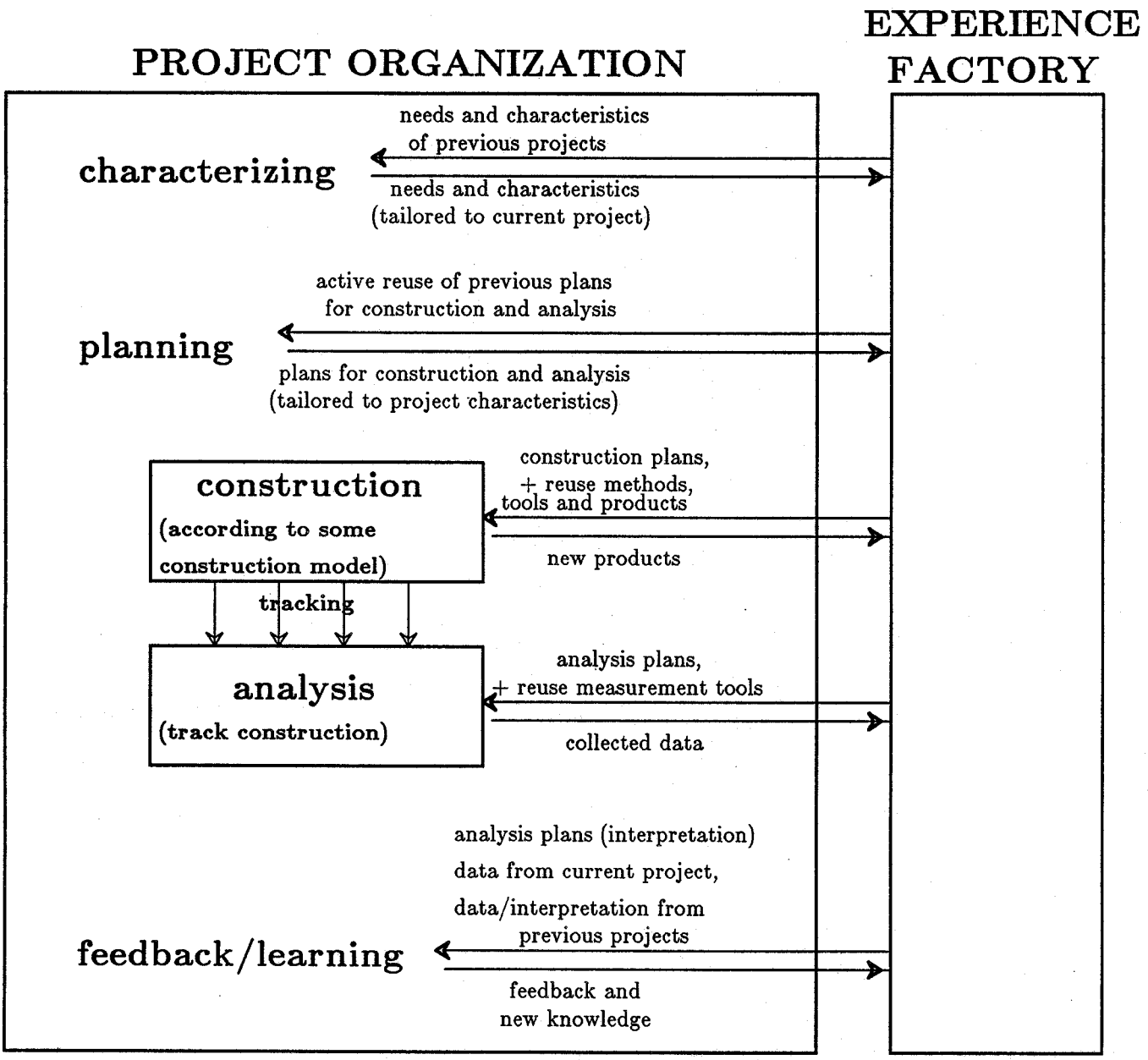

FIGURE 5: ACCESSING THE EXPERIENCE FACTORY

Project execution covers the directing and controlling activities as well as the development activities.

During the execution phase, the project proceeds using the tailored process model, methods, techniques, and tools as specified in the planning phase. It uses prior product parts, supplied by the experience base. Feedback is supplied to project management to support directing and controlling of the project. During execution, project experiences, components and data are returned to the Experience Factory and feedback is provided to the project.

At project conclusion, the overall project is analyzed and the results are fed back to the project as well as packaged and incorporated into the experience base for use on future projects.

4.2. Some Specific Activities in the Experience Factory

The Experience Factory plays several roles. It builds and maintains the experience base, it interfaces with the project in the Project Organization by providing information from the experience base and developing those elements that are requested by the project based upon its current level of expertise, e.g., tailored methods and tools and software components, and it acts as a quality assurance organization, providing feedback to the project with respect to its goals. As such it has several process models associated with it.

In building and maintaining the experience base, the Experience Factory performs the learning and reuse activities of recording, generalizing and tailoring, and formalizing. The degree to which it can perform these activities depends upon the breadth and depth of the information available and the level of technology.

It records information gathered from the various project developments. For example, it saves experiences from the projects it is monitoring, such as code modules, lessons learned on the project from the application of the constructive and analytic processes and measurement data, such as resource and defect data.

It generalizes or tailors the information that it has gathered. For example, it uses the project–specific measurement data across several projects to create baselines such as defect profiles; it develops generic packages from project specific packages or instantiates a generic package for a specific project; it refines a design technology based on the lessons learned from applying it on a specific project; it parameterizes a cost model for a project or uses data from the project to improve the estimation capability of the model.

It formalizes the information in the experience base to enhance its reuse potential. For example, it supplies code modules with their functional specifications and other appropriate documentation such as characterizing attributes, when needed; it makes more precise the steps in applying a method based upon lessons learned from its application; it builds cost models empirically based upon the data available; it develops management support systems based upon the available data and lessons learned; it builds automated support for methods.

In responding to requests from a project, it provides whatever information it has available from the experience base and the people. The level of support clearly depends upon the state of the art in the packaging of experience. The interface with the Project Organization will change over time, starting with small packets of experience and building to higher level ones.

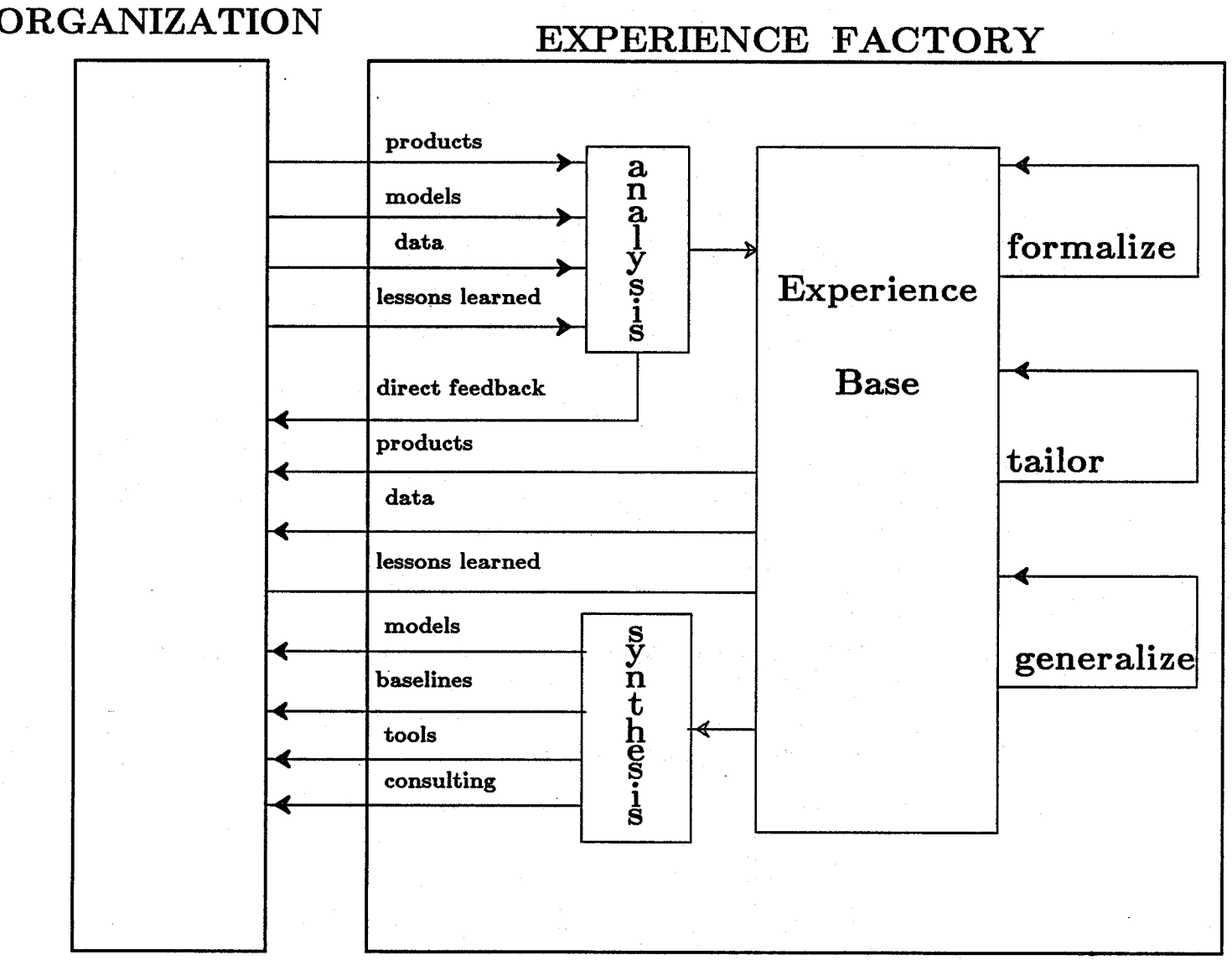

FIGURE 6: ACTIVITIES IN THE EXPERIENCE FACTORY

The actual information supplied depends upon the request and what is currently available in the experience base. For example, during characterization, it provides baselines and estimation models, and information on packaged products, such as requirements templates or code modules. General defect baselines can be tailored to the specific project by limiting the projects considered to those with the same characteristics as the current project, e.g., same application domain, same process model.

During planning it supplies GQM models and process models, methods, tools and techniques. These can be obtained directly from the experience base or tailored for the needs of the project. For example, assuming that inspections are

chosen for the project and knowing the classes of faults found in similarly classified projects, the component factory might tailor the reading technology within inspections to concentrate on locating the kinds of faults that tend to occur in this type of project. They can also provide training and consulting on the use of the methods and models.

During project execution, they can act as a contractor supplying various levels of project components. In fact from the Project Organization perspective, any component that can be well specified can be delivered by the Experience Factory. In turn, the Experience Factory can respond to the request by delivering an existing component, modifying an existing component, e.g. instantiating a generic package from the experience base, or developing the component from scratch and adding it to the experience base.

If we view quality assurance as the act of leading, teaching, and auditing the process, then it implies an organizational structure independent but interactive with the projects. (Note that this is different from quality control, which we define as the act of directing, influencing, verifying, and correcting the product, which implies a project controlled organization.) The Experience Factory is an ideal location for the quality assurance activities.

In acting as a quality assurance organization, the Experience Factory audits activities and collects the prescribed data, provides feedback to the project in real time, and offers training in the various planning, constructive, and analytic approaches. The quality assurance activities is consistent with the activities of building and maintaining the experience base and responding to requests from the Project Organization. It also provides an independent chain of command and a corporate perspective with regard to goals, data collection, process and products.

4.3. Viewing the Experience Factory as a Component Factory

As a particular dimension of the Project Organization and the Experience Factory, consider the activities of the Project Organization with regard to the development of a system and how it might use the Factory from the point of view of code development, e.g., as a Component Factory. We can view the project organization within the Project Organization as having the following activities:

Requirements Definition: The system analysts will interact with the customer to determine project requirements. It is assumed that the analysts will know the application domain and what is available in the repository for reuse. They will have access to repository information about what kinds of components are available so they can make tradeoff decisions, negotiating with the customer for function vs. price.

Initially, this negotiation will be limited since the repository will be sparsely populated. This should change over time as the repository fills with components. It should be noted that the system analyst can use Factory components for building and analyzing prototypes of the system.

Specification and Design: The requirements will be turned into a system design and specification for the required components. Those components that can be well specified can be turned over to the Experience Factory and orders will be filled for components.

Initially, the specifications will be for low level components since the Factory will begin bottom up. As time goes on and the repository builds up in terms of components, and the technology for recognizing, specifying and integrating larger pieces of systems develops, larger components can be ordered.

The Experience Factory operates according to several process models. When an order for a component arrives, it can check its repository for the appropriate component or order it externally if it is available from an outside vendor. It can develop it from scratch, using verification technology, based upon the fact that it has the specification and the component it is developing is limited in size. However, given that it has been required to deliver such a component, it can decide whether the component is of general use, from its knowledge of other projects, and can generalize or tailor the component, package it with the necessary attributes for future reuse and store it in the repository.

As an initializing activity, the Factory can analyze prior systems for reusable components and re–engineer them to seed the repository. It can develop components, so they are easy to combine, modify with respect to certain criteria and label and package appropriately.

Integration and Evaluation: The project will have the task of integrating the components into its own specified design. These integrated components might be returned to the Factory for future use. It will then evaluate the system based upon the customer requirements and deliver the system.

5. IMPLICATIONS OF THE NEW LIFE CYCLE ORGANIZATION

5.1. Implications for Corporations

One of the major problems with software development in the past has been that projects have been unable to explicitly reuse experience from prior projects or contribute to the experience base for future projects. This has been due in part to the fact that immediate project delivery goals and the more long–range goals of reuse and learning are distinct and not easily paired. Project schedule often takes precedence over the luxury of passing on learned experience.

The new life cycle organization divides the focus of software development into two separate organizations. It separates the immediate project goals from the long range learning and reuse–oriented goals. In the approach, the Project Organization can focus on the customer needs and has the advantage of access to a knowledgeable support organization in the form of the Experience Factory. The Experience Factory focuses on the organization's goals to learn and reuse. It has the advantage of accumulating experience from a large number of projects which provides it with a broader perspective than any particular project.

This organizational structure has many advantages. It should promote higher quality and productivity because of reuse and learning. It can provide better and more focused education and training for developers and provide better methods and tools for them to use.

It provides the corporation with a corporate asset in the guise of the Experience Factory. The Experience Factory contains everything the organization has learned and developed that is useful for future developments as well as an assessment of the status of corporate quality and productivity. As the Experience Factory grows in its role and assets, the corporation can learn more and more from the various experiences across the corporation.

There will be more emphasis on formalization of all parts of management and development. Formal verification becomes cost effective since the correct units will be used in many systems; it becomes more applicable since we will be applying it to smaller units, at least in the beginning, where the technology is manageable. Formal models of risk assessment can be used since the experience base should provide a broad basis for understanding and comparison.

The organizational scheme has the advantage that it can start small and expand with the growth in technology and the experience base. However, there are several issues that must be dealt with in putting this organization in place, e.g. financial and organizational.

This organization requires separate cost centers for the Project Organization and the Experience Factory. There are several models of how the funding of the Experience Factory might work. For example, it could be funded out of corporate overhead which would grow with the success of the factory or projects could be billed for factory items. The right model will depend upon the company and the organization and politics within that company.

This organization requires a careful definition of management and responsibility structures. It is clear that we do not want to create new conflicts over responsibility for problems with packaged experience.

This organization needs to be motivated and supported. Incentive and reward structures need to be developed. We will need to learn from experience gained from applying different financial and management structures.

5.2. Implications for Research

There are several implications for research based upon this organizational structure. Many of the technologies already developed for programming in the small are applicable in the factory domain. For example, verification technology is already available for factory produced components and it is necessary and cost effective because those units will be reused many times. Research activities can focus on the transfer of these technologies. Therefore, user friendly tools to support verification are needed. Based upon this formalization, we should learn more about the relevant primitives for particular application domains and how to encapsulate them.

There are research activities associated with defining and tailoring models. These include process models, methods and tools; product models of the various products and qualities of those products; and models of information, like goal generation languages, cost, resource allocation, risk, and defect prediction. Models must be defined for the Project Organization and the Experience Factory and must take into account their interface. This involves the definition of languages for defining these models and tool generators, i.e. tools that can be instantiated to support variations of a method.

There are research activities associated with generating larger product units from the Experience Factory. These include defining models of module interconnection languages that scale up, combining specifications and verifying them, and combining test plans to validate integrated components.

There are research activities associated with the building and accessing of the experience base, e.g., mechanisms for encoding lessons learned into a model, tools for generating goals and mapping them onto measures, models that permit the model to learn automatically.

5.3. Implications for Education

The organizational scheme provides a focus for many of the technologies already taught at the University and so makes much of the current education more relevant. Topics that require more emphasis are formalisms of all kinds, e.g., verification technologies, formal requirements and specification notations, formal models of measurement and management. There is a need to teach students how to develop, use and assess methods and tools and deal with access and retrieval of libraries. Reuse and learning technologies need to be made available.

There is a clear entry path for new software engineers through the Component Factory where they can develop small components under careful guidance and tool support and learn from the general experience base. As their experience grows they can be moved into any of the other higher level activities, e.g., the Project Organization, or other parts of the Experience Factory.

6. RESEARCH ACTIVITIES AT MARYLAND THAT SUPPORT THE NEW LIFE CYCLE

The paradigms and organization described in this paper offer a framework for research that focuses on the key issues for improving the software process and product in a context that permit the research to be used and experimented with in an industrial setting. Over the past dozen years, at the University of Maryland, we have been working on several research projects whose goal is to evolve to this framework.

The projects are organized into those dealing with the instantiation of the improvement paradigm in the SEL [5,46], where the concepts of the Project Organization and the Experience Factory have been evolving, the TAME project which is automating support for this framework in a formal way, and a variety of other projects which are attempting to understand, formalize and improve various process and product characteristics.

A major source of activity has been the Software Engineering Laboratory (SEL), a joint venture of the NASA Goddard Space Flight Center, the University of Maryland, and Computer Sciences Corporation. The SEL has informally acted as an Experience Factory that supports project development. The application domain is ground support software for satellites. We have been building models and supplying these models and lessons learned back to projects so they can improve their process and product. This work has been performed via experiments of various kinds, dealing with resource, defect, process, and product models.

In an attempt to better understand the environment we have used data collected during development to build various descriptive models of the SEL environment. In this way we have formalized knowledge from raw data to formal models or baselines and made the results available to the project organization for use in characterizing, planning and evaluating the project.

We have collected data on resource expenditures, applied various existing models [32,47,49,61] and eventually built and tailored models that explicitly described resource allocation in the SEL environment [2,8,10,15,30]. These are used for estimating, planning and evaluating new projects.

We have developed baselines for defects by accumulating defect data over many projects [62]. These defects a classified by phase and type. They vary with different project classifications [16]. They provide insight into the environment, support for project management and evaluation, and point to areas areas that need improvement in the process [18].

We have used various product metrics [41,45] to provide insight into the characteristics of the products being developed as well as evaluating the usefulness of these metrics for the SEL environment [6,11,26,42]. Areas of new technology that have been introduced, like Ada have generated the need for developing

new metrics to characterize new product qualities [12,40]. We have used these metrics as baselines to provide the project manager with insights as to what the problems may be with the current development [38].

With regard to process improvement, we have built descriptive and perscriptive models of processes, methods and techniques and experimented with their application. The results of our studies are formalized and reused for future projects within the limits of the technology available.

In some cases, we have performed controlled experiments in which we analyzed the effects of various methods and techniques before recommending them on actual projects. We would then perform case study experiments to evaluate the effect of the method or technology on an actual project development to assure that it scales up and is applicable to the SEL environment. For example, we ran controlled experiments on a set of structured programming methods and techniques [17], various testing and reading techniques [23,54], object oriented design in Ada [12,40] and the Cleanroom process model [59].

We then apply these approaches to projects within the project organization. We evaluate their effect there, and make recommendations, write lessons learned documents, and refine or change the models to incorporate what we have learned. In this way, the experiences gained from applying a particular model from the experience base is improved based upon the lessons learned from applying the model so that it can be used for future projects. Two case studies currently being run in the SEL, based upon controlled experiments, are the use of object oriented development in Ada [1,14,35] and the application of the Cleanroom process.

In other instances we have developed models and experimented directly on the projects. For example, we have evaluated the test methodology used for acceptance test [51] and the methods used for maintenance [55].

Parts of the data collection process have been automated for the FORTRAN environment [37, 43] and are being automated for the transition to Ada [39]. Other tools have been developed that help support the various technologies used. Parts of the evaluation process have been automated using a knowledge base to create a decision support system [50,60].

The Tame project has focused on the architecture for the measurement and evaluation processes [19]. Work has been done by using studies performed in the SEL to define the process improvement mechanisms [18]. We have devised a resource planning and feedback model that is consistent with the Improvement Paradigm [43].

The Goal/Question/Metric Paradigm has been applied in a variety of environments other than the SEL and has evolved based upon these activities [29,53,58].

We are currently working on supporting the automation of the generation of operational goals in a reasonably complete and consistent manner. A key aspect of the approach is that project personnel can generate goals that can be measured and evaluated. We are working on extending the GQM templates into a goal generation language that will aid the goal writer in articulating questions and metrics based upon the goal and the model of the object of interest. We are currently experimenting with hypertext and attribute grammar technology to develop prototypes of this automated support mechanism.

We are in various stages in the development of three measurement tools for analyzing programs in Ada and C. A source code analyzer for various syntactic metrics, such as cyclomatic complexity and software science metrics, has been developed for Ada (ASAP) [38] and C (CSAP). A structural coverage analyzer (SCA) is under development for Ada [63]. Data bindings analyzers are being developed for Ada and C based on prior versions of the tools for FORTRAN, SIMPL, and PL/C.

We have developed a set of requirements and defined a system architecture for measurement tool generations using a parser generator that retains the parse tree for further transformations [48], an enhancement of YACC and are experimenting with the prototypes of this tool generation system.

With regard to reuse, we have developed a model of a reuse support environment that can exist within the TAME framework [20]. We have applied the model to the maintenance process to show the advantages of viewing maintenance as a reuse process [7].

We are developing a model of reuse consistent with the approach presented in this paper that classifies the objects as they exist in the experience base, the reuse activities and the objects as they are reused [21]. For example, the reusable object can be classified according to the characteristics of the unit itself, its interfaces, and its context. The model recognizes the need to assess the qualities of the reusable object based upon the characteristics of the project in which it will be reused.

We are working on a language and support system that takes elementary processes and generalizes them into more complex processes. Elementary processes correspond to the "basic algorithms" used to perform small tasks, such as the addition of two atomic units. Our goal is to identify useful sets of elementary processes, and then show how they can be combined and extended to perform more complex actions (such as the addition of a stream of atomic units.) Using our language, abstract data structures may be mapped onto particular structures (e.g., the addition process for streams could be mapped onto a process for addition of arrays of numbers), and also composed with other structures (e.g., an array addition task could be composed with a division task in order to create a module for computing means.) Finally the system will package resulting processes into an acceptable language component, whether a procedure or function. Our current langague supports only functional processes, a future step is to

support the creation of data abstractions or modules.

To study the issue of code reuse, the LASER project is currently building a system that examines exising systems in order to study and extract code that can be reused to seed a component repository. The system measures the various components in the system and identifies candidate reusable components based upon their lack of complexity, reusability within the existing system, independence, etc. These candidate components are then isolated (made independent) and qualified. The qualification involves the catagorization and classification based upon a number of attributes, and the association of a functional specification with the component.

The approach expressed here provides a focus for further research issues. Some of the questions for which work has begun are:

- How can process models be formally expressed so they can be communicated, analyzed and tailored?

- How can various models be stored so they can be accessed by the GQM tool and help generate the automated collection of the appropriate measures?

- How can a specific process model be developed that satisfies the definition and storage of the prior two questions?

- How can we better capture and reuse experiences in the form of lessons learned from previous efforts?

- What other measurement data can be automatically collected?

- How could the set of measurement tools defined above be developed so that they can be tailored for various types of measures, maximizing the reuse of system components among the tools and the language independence?

- How can we classify experience so it can be appropriately reused?

- Based upon a specification, how can a component be devised quickly from elementary processes?

- How can we transform existing components to make them more independent, and measure the cost of reuse?

- How can we have confidence that the factory–provided modules will do what we want?

- How can we integrate aggregates of modules with their associated attributes so that they can be analyzed, managed, and controlled?

- How can we verify properties of aggregates of modules, not just individual modules?

- How can the test plans for components be combined to provide a test plan and oracle for aggregate application structures?

7. CONCLUSIONS

The approach expressed in this paper has evolved over years of studying and experimenting with software development and maintenance. It provides a compatable and consistent framework for both software development and software engineering research. It recognizes and takes advantage of the experimental nature of software engineering.

It allows us to understand how things are being done and where the problems are by studying the process and product in actual environments. It allows us to formalize models of the process, product and knowledge. These models can then be analyzed. They can be used to form a basis for research and at the same time provide immediate input to project development.

From a research perspective, it provides a focus for research problems based upon problems that need to be solved. It provides a framework to tie together existing pieces of research.

From a corporate perspective, the approach can be applied directly and the organization can grow and build its own experience base. It supports technology transfer in a natural way and it ties the research and development organizations closer together.

REFERENCES

[1] W. Agresti, "SEL Ada Experiment: Status and Design Experience," Proceedings of the Eleventh Annual Software Engineering Workshop, NASA Goddard Space Flight Center, Greenbelt, MD, December 1986.

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

[3] V. R. Basili, "Data Collection, Validation, and Analysis," in Tutorial on Models and Metrics for Software Management and Engineering, IEEE Catalog No. EHO–167–7, 1981, pp. 310–313.

[4] V. R. Basili, "Quantitative Evaluation of Software Engineering Methodology," Proc. of the First Pan Pacific Computer Conference, Melbourne, Australia, September 1985 [also available as Technical Report, TR–1519, Dept. of Computer Science, University of Maryland, College Park, July 1985].

[5] V. R. Basili, "Can We Measure Software Technology: Lessons Learned from 8 Years of Trying," Proceedings of the Tenth Annual Software Engineering Workshop, NASA Goddard Space Flight Center, Greenbelt, MD, December 1985.

[6] V. R. Basili, "Evaluating Software Characteristics: Assessment of Software Measures in the Software Engineering Laboratory," Proceedings of the Sixth Annual Software Engineering Workshop, NASA Goddard Space Flight Center, Greenbelt, MD, 1981.

[7] Victor R. Basili, "Software Maintenance = Reuse–Oriented Software Development," in Proc. Conference on Software Maintenance, Key–Note Address, Phoenix, AZ, October 1988 [also available as Technical Report, TR–2244, Dept. of Computer Science, University of Maryland, College Park, July 1985].

[8] V. R. Basili, J. Beane, "Can the Parr Curve help with the Manpower Distribution and Resource Estimation Problems," Journal of Systems and Software, vol. 2, no. 1, 1981, pp. 47 – 57.

[9] V. R. Basili, G. Caldiera, "Reusing Existing Software," Technical Report–2116, Institute for Advanced Computer Studies, University of Maryland, College Park, Maryland, October 1988.

[10] V. R. Basili, K. Freburger, "Programming Measurement and Estimation in the Software Engineering Laboratory," Journal of Systems and Software, vol. 2, no. 1, 1981, pp. 47–57.

[11] V. R. Basili, D. H. Hutchens, "An Empirical Study of a Syntactic Measure Family," IEEE Transactions on Software Engineering, vol. SE–9, no. 11, November 1983, pp. 664–672.

[12] V. R. Basili, E. E. Katz, "Metrics of Interest in an Ada Development," Proc. of the IEEE Computer Society Workshop on Software Engineering Technology Transfer, April 1983, pp. 22–29.

[13] V. R. Basili, E. E. Katz, "Examining the Modularity of Ada Programs," Proc. of the Joint Ada Conference, Arlington, Virginia, March 16–19, 1987.

[14] V. R. Basili, E. E. Katz, N. M. Panlilio–Yap, C. Loggia Ramsey, S. Chang, "Characterization of an Ada Software Development," IEEE Computer Magazine, September 1985, pp. 53–65.

[15] V. R. Basili, N. M. Panlilio–Yap, "Finding Relationships Between Effort and Other Variables in the SEL," IEEE COMPSAC, October 1985.

[16] V. R. Basili, B. Perricone, "Software Errors and Complexity: An Empirical Investigation," ACM Communications, vol. 27, no. 1, January 1984, pp. 45–52.

[17] V. R. Basili, R. Reiter, Jr., "A Controlled Experiment Quantitatively Comparing Software Development Approaches," IEEE Transactions on Software Engineering, vol. SE–7, no. 5, May 1981, pp. 299–320.

[18] 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.

[19] 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.

[20] V. R. Basili, H. D. Rombach, "Software Reuse: A Comprehensive Framework," CS–TR–2158, Department of Computer Science, University of Maryland, College Park, Maryland.

[21] 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.

[22] V. R. Basili, R. W. Selby, Jr., "Data Collection and Analysis in Software Research and Management," Proc. of the American Statistical Association and Biomeasure Society Joint Statistical Meetings, Philadelphia, PA, August 13–16, 1984.

[23] Victor R. Basili, R. W. Selby, "Comparing the Effectiveness of Software Testing Strategies," IEEE Transactions on Software Engineering, Vol. SE–13, No. 12, December 1987, pp. 1278–1296.

[24] V. R. Basili, R. W. Selby, Jr., "Calculation and Use of an Environment's Characteristic Software Metric Set," Proceedings of the Eighth International Conference on Software Engineering, London, UK, August 1985.

[25] 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.

[26] V. R. Basili, R. W. Selby, and T.–Y. Phillips, "Metric Analysis and Data Validation Across Fortran Projects," IEEE Transactions on Software Engineering, vol. SE–9, no. 6, November 1983, pp. 652–663.

[27] V. R. Basili, A. J. Turner, "Iterative Enhancement: A Practical Technique for Software Development," IEEE Transactions on Software Engineering, vol. SE–1, no. 4, December 1975.

[28] V. R. Basili, D. M. Weiss, "A Methodology for Collecting Valid Software Engineering Data," IEEE Transactions on Software Engineering, vol. SE–10, no.6, November 1984, pp. 728–738.

[29] V. R. Basili, D. M. Weiss, "Evaluation of a Software Requirements Document by Analysis of Change Data," Proceedings of the Fifth International

Conference on Software Engineering, San Diego, USA, March 1981, pp. 314–323.

[30] V. R. Basili, M. V. Zelkowitz, "Analyzing Medium Scale Software Development," Proceedings of the Third International Conference on Software Engineering, Atlanta, Georgia, USA, May 1978, pp. 116–123.

[31] B. W. Boehm, "Software Engineering," IEEE Transactions on Computers, vol. C–25, no. 12, December 1976, pp. 1226–1241.

[32] B. W. Boehm, "Software Engineering Economics," Prentice–Hall, Englewood Cliffs, NJ, 1981.

[33] B. W. Boehm, "A Spiral Model of Software Development and Enhancement," ACM Software Engineering Notes, vol. 11, no. 4, August 1986, pp. 22–42.

[34] B. W. Boehm, J. R. Brown, and M. Lipow, "Quantitative Evaluation of Software Quality," Proceedings of the Second International Conference on Software Engineering, 1976, pp. 592–605.

[35] C. Brophy, W. Agresti, and V. R. Basili, "Lessons Learned in Use of Ada Oriented Design Methods," Proc. of the Joint Ada Conference, Arlington, Virginia, March 16–19, 1987.

[36] W. J. Decker, W. A. Taylor, "Fortran Static Source Code Analyzer Program (SAP)," Technical Report SEL–82–002, NASA Goddard Space Flight Center, August 1982.

[37] C. W. Doerflinger, V. R. Basili, "Monitoring Software Development Through Dynamic Variables," IEEE Transactions on Software Engineering, vol. SE–11, no. 9, September 1985, pp. 978–985.

[38] D. L. Doubleday, "ASAP: An Ada Static Source Code Analyzer Program," Technical Report, TR–1895, Deptartment of Computer Science, University of Maryland, College Park, August 1987.

[39] M. Dyer, "Cleanroom Software Development Method," IBM Federal Systems Division, Bethesda, Maryland, October 14, 1982.

[40] J. Gannon, E. E. Katz, and V. R. Basili, "Measures for Ada Packages: An Initial Study," Communications of the ACM, vol. 29, no. 7, July 1986, pp. 616–623.

[41] M. H. Halstead, "Elements of Software Science," Elsevier North–Holland, New York, 1977.

[42] D. H. Hutchens, V. R. Basili, "System Structure Analysis: Clustering with

Data Bindings," IEEE Transactions on Software Engineering, August 1985, pp. 749–757.

[43] D. R. Jefferey, V. R. Basili, "Validating the TAME Resource Data Model," Proceedings of the Tenth International Conference on Software Engineering, Singapore, April, 1988, pp. 187–201.

[44] E. E. Katz, H. D. Rombach, and V. R. Basili, "Structure and Maintainability of Ada Programs: Can We Measure the Differences?," Proc. of the Ninth Minnowbrook Workshop on Software Performance Evaluation, Blue Mountain Lake, New York, August 5–8, 1986.

[45] T. J. McCabe, "A Complexity Measure," IEEE Transactions on Software Engineering, December 1976, pp. 308–320.

[46] F. E. McGarry, "Recent SEL Studies," Proceedings of the Tenth Annual Software Engineering Workshop, NASA Goddard Space Flight Center, December 1985.

[47] F. N. Parr, "An Alternative to the Rayleigh Curve Model for Software Development Effort," IEEE Transactions on Software Engineering, vol. SE–6, no. 3, March 1980.

[48] J. Purtilo and J. Callahan, "Parse Tree Annotations", Communications of the ACM, to appear.

[49] L. Putnam, "A General Empirical Solution to the Macro Software Sizing and Estimating Problem," IEEE Transactions on Software Engineering, vol. SE–4, no. 4, April 1978, pp. 345–361.

[50] C. Loggia–Ramsey, V. R. Basili, "An Evaluation of Expert Systems for Software Engineering Management," IEEE Transactions on Software Engineering, Vol. 15, no. 6, June 1989, pp. 747–7597.

[51] J. Ramsey, V. R. Basili, "Analyzing the Test Process Using Structural Coverage," Proceedings of the Eighth International Conference on Software Engineering, London, UK, August 1985.

[52] H. D. Rombach, "Software Design Metrics for Maintenance," Proceedings of the Ninth Annual Software Engineering Workshop, NASA Goddard Space Flight Center, Greenbelt, MD, November 1984.

[53] H. D. Rombach, V. R. Basili, "A Quantitative Assessment of Software Maintenance: An Industrial Case Study," Conference on Software Maintenance, Austin, Texas, September 1987.

[54] H. D. Rombach, V. R. Basili, and R. W. Selby, Jr., "The Role of Code

Reading in the Software Life Cycle," Proc. of the Ninth Minnowbrook Workshop on Software Performance Evaluation, Blue Mountain Lake, New York, August 5–8, 1986.

[55] H. D. Rombach, B. T. Ulery, "Establishing a Measurement–Based Maintenance Environment Program: Lessons Learned in the SEL", Proceedings of the IEEE Conference on Software Maintenance, Miami Beach, October, 1989.

[56] W. W. Royce, "Managing the Development of Large Software Systems: Concepts and Techniques," Proceedings of the WESCON, August 1970.

[57] R. W. Selby, Jr., "Incorporating Metrics into a Software Environment," Proceedings of the Joint Ada Conference, Arlington, VA, March 16–19, 1987, pp. 326–333.

[58] R. W. Selby, Jr., V. R. Basili, "Analyzing Error–Prone System Coupling and Cohesion," Technical Report TR–88–46, Institute for Advanced Computer Studies, University of Maryland, College Park, Maryland, June 1988.

[59] R. W. Selby, Jr., V. R. Basili, and T. Baker, "CLEANROOM Software Development: An Empirical Evaluation," IEEE Transactions on Software Engineering, Vol. 13 no. 9, September, 1987, pp. 1027–1037.

[60] J. D. Valett, "The Dynamic Management Information Tool (DYNAMITE):Analysis of the Prototype, Requirements and Operational Scenarios," M.Sc. Thesis, University of Maryland, 1987.

[61] C. E. Walston, C. P. Felix, "A Method of Programming Measurement and Estimation," IBM Systems Journal, vol. 16, no. 1, 1977, pp. 54–73.

[62] D. M. Weiss, V. R. Basili, "Evaluating Software Development by Analysis of Changes: Some Data from the Software Engineering Laboratory," IEEE Transactions on Software Engineering, vol. SE–11, no. 2, February 1985, pp. 157–168.

[63] L. Wu, V. R. Basili, and K. Reed, "A Structure Coverage Tool for Ada Software Systems," Proc. of the Joint Ada Conference, Arlington, Virginia, March 16–19, 1987.

[64] M. Zelkowitz, R. Yeh, R. Hamlet, J. Gannon, and V. R. Basili, "Software Engineering Practices in the U.S. and Japan," IEEE Computer Magazine, June 1984, pp. 57–66.