The TAME Project: Towards Improvement-Oriented Software Environments

VICTOR R. BASILI, SENIOR MEMBER, IEEE, AND H. DIETER ROMBACH

Abstract—**Experience from a dozen years of analyzing software engineering processes and products is summarized as a set of software engineering and measurement principles that argue for software engineering process models that integrate sound planning and analysis into the construction process.**

In the TAME (Tailoring A Measurement Environment) project at the University of Maryland we have developed such an improvement-oriented software engineering process model that uses the goal/question/metric paradigm to integrate the constructive and analytic aspects of software development. The model provides a mechanism for formalizing the characterization and planning tasks, controlling and improving projects based on quantitative analysis, learning in a deeper and more systematic way about the software process and product, and feeding the appropriate experience back into the current and future projects.

The TAME system is an instantiation of the TAME software engineering process model as an ISEE (Integrated Software Engineering Environment). The first in a series of TAME system prototypes has been developed. An assessment of experience with this first limited prototype is presented including a reassessment of its initial architecture. The long-term goal of this building effort is to develop a better understanding of appropriate ISEE architectures that optimally support the improvement-oriented TAME software engineering process model.

Index Terms—**Characterization, execution, experience, feedback, formalizing, goal/question/metric paradigm, improvement paradigm, integrated software engineering environments, integration of construction and analysis, learning, measurement, planning, quantitative analysis, software engineering process models, tailoring, TAME project, TAME system.**

I. Introduction

EXPERIENCE from a dozen years of analyzing software engineering processes and products is summarized as a set of ten *software engineering* and fourteen *measurement principles*. These principles imply the need for software engineering process models that integrate sound planning and analysis into the construction process.

Software processes based upon such *improvement-oriented software engineering process models* need to be *tailorable* and *tractable*. The tailorability of a process is the characteristic that allows it to be altered or adapted to suit a set of special needs or purposes [64]. The software engineering process requires tailorability because the overall project execution model (life cycle model), methods and tools need to be altered or adapted for the specific project environment and the overall organization. The tractability of a process is the characteristic that allows it to be easily planned, taught, managed, executed, or controlled [64]. Each software engineering process requires tractability because it needs to be planned, the various planned activities of the process need to be communicated to the entire project personnel, and the process needs to be managed, executed, and controlled according to these plans. Sound tailoring and tracking require *top-down measurement* (measurement based upon operationally defined goals). The goal of a *software engineering environment (SEE)* should be to support such tailorable and tractable software engineering process models by automating as much of them as possible.

In the *TAME (Tailoring A Measurement Environment) project* at the University of Maryland we have developed an improvement-oriented software engineering process model. The *TAME system* is an instantiation of this TAME software engineering process model as an ISEE (Integrated SEE).

It seems appropriate at this point to clarify some of the important terms that will be used in this paper. The term *engineering* comprises both development and maintenance. A software engineering *project* is embedded in some *project environment* (characterized by personnel, type of application, etc.) and within some *organization* (e.g., NASA, IBM). Software engineering within such a project environment or organization is conducted according to an overall software engineering *process model* (one of which will be introduced in Section II-B-3). Each individual software project in the context of such a software engineering process model is exeucted according to some *execution model* (e.g., waterfall model [28], [58], iterative enhancement model [24], spiral model [30]) supplemented by *techniques (methods, tools)*. Each specific instance of (a part of) an execution model together with its supplementing methods and tools is referred to as *execution process* (including the construction as well as the analysis process). In addition, the term *process* is frequently used as a generic term for various kinds of activities. We distinguish between *constructive* and *analytic* methods and tools. Whereas constructive methods and tools are concerned with building products, analytic

Manuscript received January 15, 1988. This work was supported in part by NASA under Grant NSG-5123, the Air Force Office of Scientific Research under Grant F49620-87-0130, and the Office of Naval Research under Grant N00014-85-K-0633 to the University of Maryland. Computer time was provided in part through the facilities of the Computer Science Center of the University of Maryland.

The authors are with the Department of Computer Science and the Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742.

IEEE Log Number 8820962.

method and tools are concerned with analyzing the constructive process and the resulting products. The body of experience accumulated within a project environment or organization is referred to as *experience base*. There exist at least three levels of formalism of such experience bases: *database* (data being individual products or processes), *information base* (information being data viewed through some superimposed structure), and knowledge base (knowledge implying the ability to derive new insights via deduction rules). The project personnel are categorized as either *engineers* (e.g., designers, coders, testers) or *managers*.

This paper is structured into a presentation and discussion of the improvement-oriented software engineering process model underlying the TAME project (Section II), its automated support by the TAME system (Section III), and the first TAME system prototype (Section IV). In the first part of this paper we list the empirically derived lessons learned (Section II-A) in the form of software engineering principles (Section II-A-1), measurement principles (Section II-A-2), and motivate the TAME project by stating several implications derived from those principles (Section II-A-3). The TAME project (Section II-B) is presented in terms of the improvement paradigm (Section II-B-1), the goal/question/metric paradigm as a mechanism for formalizing the improvement paradigm (Section II-B-2), and the TAME project model as an instantiation of both paradigms (Section II-B-3). In the second part of this paper we introduce the TAME system as an approach to automatically supporting the TAME software engineering process model (Section III). The TAME system is presented in terms of its requirements (Section III-A) and architecture (Section III-B). In the third part of this paper, we introduce the first TAME prototype (Section IV) with respect to its functionality and our first experiences with it.

II. Software Engineering Process

Our experience from measuring and evaluating software engineering processes and products in a variety of project environments has been summarized in the form of lessons learned (Section II-A). Based upon this experience the TAME project has produced an improvement-oriented process model (Section II-B).

A. Lessons Learned from Past Experience

We have formulated our experience as a set of software engineering principles (Section II-A-1) and measurement principles (Section II-A-2). Based upon these principles a number of implications for sound software engineering process models have been derived (Section II-A-3).

1) Software Engineering Principles: The first five software engineering principles address the need for developing quality *a priori* by introducing engineering discipline into the field of software engineering:

(P1) We need to clearly distinguish between the role of constructive and analytic activities. Only improved construction processes will result in higher quality software. Quality cannot be tested or inspected into software. Analytic processes (e.g., quality assurance) cannot serve as a substitute for constructive processes but will provide control of the constructive processes [27], [37], [61].

(P2) We need to formalize the planning of the construction process in order to develop quality *a priori* [3], [16], [19], [25]. Without such plans the trial and error approach can hardly be avoided.

(P3) We need to formalize the analysis and improvement of construction processes and products in order to guarantee an organized approach to software engineering [3], [25].

(P4) Engineering methods require analysis to determine whether they are being performed appropriately, if at all. This is especially important because most of these methods are heuristic rather than formal [42], [49], [66].

(P5) Software engineers and managers need real-time feedback in order to improve the construction processes and products of the ongoing project. The organization needs post-mortem feedback in order to improve the construction processes and products for future projects [66].

The remaining five software engineering principles address the need for tailoring of planning and analysis processes due to changing needs form project to project and environment to environment:

(P6) All project environments and products are different in some way [2], [66]. These differences must be made explicit and taken into account in the software execution processes and in the product quality goals [3], [16], [19], [25].

(P7) There are many execution models for software engineering. Each execution model needs to be tailored to the organization and project needs and characteristics [2], [13], [16], [66].

(P8) We need to formalize the tailoring of processes toward the quality and productivity goals of the project and the characteristics of the project environment and the organization [16]. It is not easy to apply abstractly defined methods to specific environments.

(P9) This need for tailoring does not mean starting from scratch each time. We need to reuse experience, but only after tailoring it to the project [1], [2], [6], [7], [18], [32].

(P10) Because of the constant need for tailoring, management control is crucial and must be flexible. Management needs must be supported in this software engineering process.

A more detailed discussion of these software engineering principles is contained in [17].

2) Software Measurement Principles: The first four measurement principles address the purpose of the measurement process, i.e., why should we measure, what should we measure, for whom should we measure:

(M1) Measurement is an ideal mechanism for characterizing, evaluating, predicting, and providing motivation for the various aspects of software construction processes and products [3], [4], [9], [16], [21], [25], [48], [56], [57]. It is a common mechanism for relating these multiple aspects.

(M2) Measurements must be taken on both the soft-

ware processes and the various software products [1], [5], [14], [29], [38], [40], [42]-[44], [47], [54]-[56], [65], [66]. Improving a product requires understanding both the product and its construction processes.

(M3) There are a variety of uses for measurement. The purpose of measurement should be clearly stated. We can use measurement to examine cost, effectiveness, reliability, correctness, maintainability, efficiency, user friendliness, etc. [8]-[10], [13], [14], [16], [20], [23], [25], [41], [53], [57], [61].

(M4) Measurement needs to be viewed from the appropriate perspective. The corporation, the manager, the developer, the customer's organization and the user each view the product and the process from different perspectives. Thus they may want to know different things about the project and to different levels of detail [3], [16], [19], [25], [66].

The remaining ten measurement principles address metrics and the overall measurement process. The first two principles address characteristics of metrics (i.e., what kinds of metrics, how many are needed), while the latter eight address characteristics of the measurement process (i.e., what should the measurement process look like, how do we support characterization, planning, construction, and learning and feedback):

(M5) Subjective as well as objective metrics are required. Many process, product and environment aspects can be characterized by objective metrics (e.g., product complexity, number of defects or effort related to processes). Other aspects cannot be characterized objectively yet (e.g., experience of personnel, type of application, understandability of processes and products); but they can at least be categorized on a quantitative (nominal) scale to a reasonable degree of accuracy [4], [5], [16], [48], [56].

(M6) Most aspects of software processes and products are too complicated to be captured by a single metric. For both definition and interpretation purposes, a set of metrics (a metric vector) that frame the purpose for measurement needs to be defined [9].

(M7) The development and maintenance environments must be prepared for measurement and analysis. Planning is required and needs to be carefully integrated into the overall software engineering process model. This planning process must take into account the experimental design appropriate for the situation [3], [14], [19], [22], [66].

(M8) We cannot just use models and metrics from other environments as defined. Because of the differences among execution models (principle P7), the models and metrics must be tailored for the environment in which they will be applied and checked for validity in that environment [2], [6]-[8], [12], [23], [31], [40], [47], [50], [51], [62].

(M9) The measurement process must be top-down rather than bottom-up in order to define a set of operational goals, specify the appropriate metrics, permit valid contextual interpretation and analysis, and provide feedback for tailorability and tractability [3], [16], [19], [25].

(M10) For each environment there exists a characteristic set of metrics that provides the needed information for definition and interpretation purposes [21].

(M11) Multiple mechanisms are needed for data collection and validation. The nature of the data to be collected (principle M5) determines the appropriate mechanisms [4], [25], [48], e.g., manually via forms or interviews, or automatically via analyzers.

(M12) In order to evaluate and compare projects and to develop models we need a historical experience base. This experience base should characterize the local environment [4], [13], [25], [34], [44], [48].

(M13) Metrics must be associated with interpretations, but these interpretations must be given in context [3], [16], [19], [25], [34], [56].

(M14) The experience base should evolve from a database into a knowledge base (supported by an expert system) to formalize the reuse of experience [11], [14].

A more detailed discussion of these measurement principles is contained in [17].

3) Implications: Clearly this set of principles is not complete. However, these principles provide empirically derived insight into the limitations of traditional process models. We will give some of the implications of these principles with respect to the components that need to be included in software process models, essential characteristics of these components, the interaction of these components, and the needed automated support. Although there is a relationship between almost all principles and the derived implications, we have referenced for each implication only those principles that are related most directly.

Based upon our set of principles it is clear that we need to *better understand* the software construction process and product (e.g., principles P1, P4, P6, M2, M5, M6, M8, M9, M10, M12). Such an understanding will allow us to *plan* what we need to do and improve over our current practices (e.g., principles P1, P2, P3, P7, P8, M3, M4, M7, M9, M14). To *make those plans operational*, we need to specify how we are going to affect the construction processes and their analysis (e.g., principles P1, P2, P3, P4, P7, P8, M7, M8, M9, M14). The *execution* of these prescribed plans involves the *construction* of products and the *analysis* of the constructive processes and resulting products (e.g., principles P1, P7).

All these implications need to be integrated in such a way that they allow for sound *learning and feedback* so that we can improve the software execution processes and products (e.g., principles P1, P3, P4, P5, P9, P10, M3, M4, M9, M12, M13, M14). This interaction requires the integration of the constructive and analytic aspects of the software engineering process model (e.g., principles P2, M7, M9).

The components and their interactions need to be formalized so they can be supported properly by an *ISEE*

(e.g., principles P2, P3, P8, P9, M9). This formalization must include a *structuring of the body of experience* so that characterization, planning, learning, feedback, and improvement can take place (e.g., principles P2, P3, P8, P9, M9). An ideal mechanism for supporting all of these components and their interactions is *quantitative analysis* (e.g., principles P3, P4, M1, M2, M5, M6, M8, M9, M10, M11, M13).

B. A Process Model: The TAME Project

The TAME (Tailoring A Measurement Environment) project at the University of Maryland has produced a software engineering process model (Section II-B-3) based upon our empirically derived lessons learned. This software engineering process model is based upon the improvement (Section II-B-1) and goal/question/metric paradigms (Section II-B-2).

1) Improvement Paradigm: The improvement paradigm for software engineering processes reflects the implications stated in Section II-A-3. It consists of six major steps [3]:

(I1) Characterize the current project environment.

(I2) Set up goals and refine them into quantifiable questions and metrics for successful project performance and improvement over previous project performances.

(I3) Choose the appropriate software project execution model for this project and supporting methods and tools.

(I4) Execute the chosen processes and construct the products, collect the prescribed data, validate it, and provide feedback in real-time.

(I5) Analyze the data to evaluate the current practices, determine problems, record the findings, and make recommendations for improvement.

(I6) Proceed to Step I1 to start the next project, armed with the experience gained from this and previous projects.

This paradigm is aimed at providing a basis for corporate learning and improvement. Improvement is only possible if we a) understand what the current status of our environment is (step I1), b) state precise improvement goals for the particular project and quantify them for the purpose of control (step I2), c) choose the appropriate process execution models, methods, and tools in order to achieve these improvement goals (step I3), execute and monitor the project performance thoroughly (step I4), and assess it (step I5). Based upon the assessment results we can provide feedback into the ongoing project or into the planning step of future projects (steps I5 and I6).

2) Goal/Question/Metric Paradigm: The goal/question/metric (GQM) paradigm is intended as a mechanism for formalizing the characterization, planning, construction, analysis, learning and feedback tasks. It represents a systematic approach for setting project goals (tailored to the specific needs of an organization) and defining them in an operational and tractable way. Goals are refined into a set of quantifiable questions that specify metrics. This paradigm also supports the analysis and integration of metrics in the context of the questions and the original goal. Feedback and learning are then performed in the context of the GQM paradigm.

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. These templates and guidelines reflect our experience from having applied the GQM paradigm in a variety of environments (e.g., NASA [4], [17], [48], IBM [60], AT&T, Burroughs [56], and Motorola). We received additional feedback from Hewlett Packard where the GQM paradigm has been used without our direct assistance [39]. It needs to be stressed that we do not claim that these templates and guidelines are complete; they will most likely change over time as our experience grows. 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, context), defining the quality perspective of interest (e.g., reliability, user friendliness), and providing feedback from the particular quality perspective. Process-related questions are formulated for the purpose of defining the process (quality of use, domain of use), defining the quality perspective of interest (e.g., reduction of defects, cost effectiveness of use), and providing feedback from the particular quality perspective.

- **Templates/Guidelines for Goal Definition:**

Purpose: To (characterize, evaluate, predict, motivate, etc.) the (process, product, model, metric, etc.) in order to (understand, assess, manage, engineer, learn, improve, etc.) it.

Example: To evaluate the system testing methodology in order to improve it.

Perspective: Examine the (cost, effectiveness, correctness, defects, changes, product metrics, reliability, etc.) from the point of view of the (developer, manager, customer, corporate perspective, etc.)

Example: Examine the effectiveness from the developer's point of view.

Environment: The environment consists of the following: process factors, people factors, problem factors, methods, tools, constraints, etc.

Example: The product is an operating system that must fit on a PC, etc.

- **Guidelines for Product-Related Questions:**

For each product under study there are three major subgoals that need to be addressed: 1) definition of the product, 2) definition of the quality perspectives of interest, and 3) feedback related to the quality perspectives of interest.

Definition of the product includes questions related to *physical attributes* (a quantitative characterization of the product in terms of physical attributes such as size, com-

plexity, etc.), *cost* (a quantitative characterization of the resources expended related to this product in terms of effort, computer time, etc.), *changes and defects* (a quantitative characterization of the errors, faults, failures, adaptations, and enhancements related to this product), and *context* (a quantitative characterization of the customer community using this product and their operational profiles).

Quality perspectives of interest includes, for each quality perspective of interest (e.g., reliability, user friendliness), questions related to the *major model(s) used* (a quantitative specification of the quality perspective of interest), the *validity of the model for the particular environment* (an analysis of the appropriateness of the model for the particular project environment), the *validity of the data collected* (an analysis of the quality of data), the *model effectiveness* (a quantitative characterization of the quality of the results produced according to this model), and a *substantiation of the model* (a discussion of whether the results are reasonable from various perspectives).

Feedback includes questions related to *improving the product relative to the quality perspective of interest* (a quantitative characterization of the product quality, major problems regarding the quality perspective of interest, and suggestions for improvement during the ongoing project as well as during future projects).

- **Guidelines for Process-Related Questions**

For each process under study, there are three major subgoals that need to be addressed: 1) definition of the process, 2) definition of the quality perspectives of interest, and 3) feedback from using this process relative to the quality perspective of interest.

Definition of the process includes questions related to the *quality of use* (a quantitative characterization of the process and an assessment of how well it is performed), and the *domain of use* (a quantitative characterization of the object to which the process is applied and an analysis of the process performer's knowledge concering this object).

Quality perspectives of interest follows a pattern similar to the corresponding product-oriented subgoal including, for each quality perspective of interest (e.g., reduction of defects, cost effectiveness), questions related to the *major model(s) used*, and *validity of the model for the particular environment*, the *validity of the data collected*, the *model effectiveness* and the *substantiation of the model*).

Feedback follows a pattern similar to the corresponding product-oriented subgoal.

- **Guidelines for Metrics, Data Collection, and Interpretation:**

The choice of metrics is determined by the quantifiable questions. The guidelines for questions acknowledge the need for generally more than one metric (principle M6), for objective and subjective metrics (principle M5), and for associating interpretations with metrics (principle M13). The actual GQM models generated from these templates and guidelines will differ from project to project and organization to organization (principle M6). This reflects their being tailored for the different needs in different projects and organizations (principle M4). Depending on the type of each metric, we choose the appropriate mechansims for data collection and validation (principle M11). As goals, questions and metrics provide for tractability of the (top-down) definitional quantification process, they also provide for the interpretation context (bottom-up). This integration of definition with interpretation allows for the interpretation process to be tailored to the specific needs of an environment (principle M8).

3) Improvement-Oriented Process Model: The TAME software engineering process model is an instantiation of the improvement paradigm. The GQM paradigm provides the necessary integration of the individual components of this model. The TAME software engineering process model explicitly includes components for (C1) the characterization of the current status of a project environment, (C2) the planning for improvement integrated into the execution of projects, (C3) the execution of the construction and analysis of projects according to the project plans, and (C4) the recording of experience into an experience base. The learning and feedback mechanism (C5) is distributed throughout the model within and across the components as information flows from one component to another. Each of these tasks must be dealt with from a constructive and analytic perspective. Fig. 1 contains a graphical representation of the improvement-oriented TAME process model. The relationships (arcs) among process model components in Fig. 1 represent information flow.

(C1) Characterization of the current environment is required to understand the various factors that influence the current project environment. This task is important in order to define a starting point for improvement. Without knowing where we are, we will not be able to judge whether we are improving in our present project. We distinguish between the constructive and analytic aspects of the characterization task to emphasize that we not only state the environmental factors but analyze them to the degree possible based upon data and other forms of information from prior projects. This characterization task needs to be formalized.

(C2) Planning is required to understand the project goals, execution needs, and project focus for learning and feedback. This task is essential for disciplined software project execution (i.e., executing projects according to precise specifications of processes and products). It provides the basis for improvement relative to the current status determined during characterization. In the planning task, we distinguish between the constructive and analytic as well as the "what" and "how" aspects of planning. Based upon the GQM paradigm all these aspects are highly interdependent and performed as a single task. The development of quantitatively analyzable goals is an iterative process. However, we formulate the four planning as-

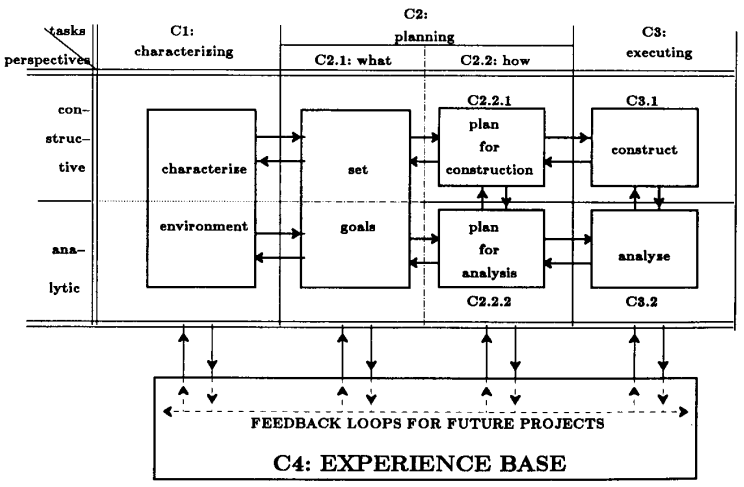

Fig. 1. The improvement-oriented TAME software process model.

pects as four separate components to emphasize the differences between creating plans for development and making those plans analyzable, as well as between stating what it is you want to accomplish and stating how you plan to tailor the processes and metrics to do it.

(C2.1) "What" Planning deals with choosing, assigning priorities, and operationally defining, to the degree possible, the project goals from the constructive and analytic perspectives. The actual goal setting is an instantiation of the front-end of the GQM paradigm (the templates/guidelines for goal definition). The constructive perspective addresses the definition of project goals such as on-time delivery, the appropriate functionality to satisfy the user, and the analysis of the execution processes we are applying. Some of these goals might be stated as improvement goals over the current state-of-the-practice as characterized in component C1. These goals should be prioritized and operationally defined to the extent possible without having chosen the particular construction models, methods and tools yet. The analytic perspective addresses analysis procedures for monitoring and controlling whether the goals are met. This analytic goal perspective should prescribe the necessary learning and feedback paths. It should be operationally defined to the extent allowed by the degree of precision of the constructive goal perspective.

(C2.2) "How" Planning is based upon the results from the "what" planning (providing for the purpose and perspective of a goal definition according to the GQM paradigm front-end) and the characterization of the environment (providing for the environment part of a goal definition according to the GQM paradigm front-end). The "how" planning involves the choice of an appropriately tailored execution model, methods and tools that permit the building of the system in such a way that we can analyze whether we are achieving our stated goals. The particular choice of construction processes, methods and tools (component C2.2.1) goes hand in hand with fine-tuning the analysis procedures derived during the analytic perspective of the "what" planning (component C2.2.2).

(C2.2.1) Planning for construction includes choosing the appropriate execution model, methods and tools to fulfill the project goals. It should be clear that effective planning for construction depends on well-defined project goals from both the constructive and analytic perspective (component C2.1).

(C2.2.2) Planning for analysis addresses the fine-tuning of the operational definition of the analytic goal perspective (derived as part of component C2.1) towards the specific choices made during planning for construction (C2.2.1). The actual planning for analysis is an instantiation of the back-end of the GQM paradigm; details need to be filled in (e.g., quantifiable questions, metrics) based upon the specific methods and tools chosen.

(C3) Execution must integrate the construction (component C3.1) with the analysis (component C3.2). Analysis (including measurement) cannot be an add-on but must be part of the execution process and drive the construction. The execution plans derived during the planning task are supposed to provide for the required integration of construction and analysis.

(C4) The Experience Base includes the entire body of experience that is actively available to the project. We can characterize this experience according to the following dimensions: a) the degree of precision/detail, and b) the degree to which it is tailored to meet the specific needs of the project (context). The precision/detail dimension involves the level of detail of the experimental design and the level and quality of data collected. On one end of the spectrum we have detailed objective quantitative data that allows us to build mathematically tractable models. On the other end of the spectrum we have interviews and qualitative information that provide guidelines and "lessons learned documents", and permit the better formu-

lation of goals and questions. The level of precision and detail affects our level of confidence in the results of the experiment as well as the cost of the data collection process. Clearly priorities play an important role here. The context dimension involves whether the focus is to learn about the specific project, projects within a specific application domain or general truths about the software process or product (requires the incorporation of formalized experience from prior projects into the experience base). Movement across the context dimension assumes an 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 environments.

(C5) Learning and Feedback are integrated into the TAME process model in various ways. They are based upon the experimental model for learning consisting of a set of steps, starting with focused objectives, which are turned into specific hypotheses, followed by running experiments to validate the hypotheses in the appropriate environment. The model is iterative; as we learn from experimentation, we are better able to state our focused objectives and we change and refine our hypotheses.

This model of learning is incorporated into the GQM paradigm where the focused objectives are expressed as goals, the hypotheses are expressed as questions written to the degree of formalism required, and the experimental environment is the project, a set of projects in the same domain, or a corporation representing a general environment. Clearly the GQM paradigm is also iterative.

The feedback process helps generate the goals to influence one or more of the components in the process model, e.g., the characterization of the environment, or the analysis of the construction processes or products. The level of confidence we have in feeding back the experience to a project or a corporate environment depends upon the precision/detail level of the experience base (component C4) and the generality of the experimental environment in which it was gathered.

The learning and feedback process appears in the model as the integration of all the components and their interactions as they are driven by the improvement and GQM paradigms. The feedback process can be channeled to the various components of the current project and to the corporate experience base for use in future projects.

Most traditional software engineering process models address only a subset of the individual components of this model; in many cases they cover just the constructive aspects of characterization (component C1), "how" planning (component C2.2.1), and execution (component C3.1). More recently developed software engineering process models address the constructive aspect of execution (component C3.1) in more sophisticated ways (e.g., new process models [24],[30], [49], combine various process dimensions such as technical, managerial, contractual [36], or provide more flexibility as far as the use of methods and tools is concerned, for example via the automated generation of tools [45], [63]), or they add methods and tools for choosing the analytical processes, methods, and tools (component C3.2.2) as well as actually performing analysis (component C3.2) [52], [59]. However, all these process models have in common the lack of completely integrating all their individual components in a systematic way that would permit sound learning and feedback for the purpose of project control and improvement of corporate experience.

III. Automated Support through ISEEs: The TAME System

The goal of an Integrated Software Engineering Environment (ISEE) is to effectively support the improvement-oriented software engineering process model described in Section II-B-3. An ISEE must support all the model components (characterization, planning, execution, and the experience base), all the local interactions between model components, the integration, and formalization of the GQM paradigm, and the necessary transitions between the context and precision/detail dimension boundaries in the experience base. Supporting the transitions along the experience base dimensions is needed in order to allow for sound learning and feedback as outlined in Section II-B-3 (component C5).

The TAME system will automate as many of the components, interactions between components and supporting mechanisms of the TAME process model as possible. The TAME system development activities will concentrate on all but the construction component (component C3.1) with the eventual goal of interfacing with constructive SEEs. In this section we present the requirements and the initial architecture for the TAME system.

A. Requirements

The requirements for the TAME system can be derived from Section II-B-3 in a natural way. These requirements can be divided into external requirements (defined by and of obvious interest to the TAME system user) and internal requirements (defined by the TAME design team and required to support the external requirements properly).

The first five (external) requirements include support for the characterization and planning components of the TAME model by automating an instantiation of the GQM paradigm, for the analysis component by automating data collection, data validation and analysis, and the learning and feedback component by automating interpretation and organizational learning. We will list for each external TAME system requirement the TAME process model components of Section II-B-3 from which it has been derived.

External TAME requirements:

(R1) A mechanism for defining the constructive and analytic aspects of project goals in an operational and quantifiable way (derived from components C1, C2.1, C2.2.2, C3.2).

We use the GQM paradigm and its templates for defin-

ing goals operationally and refining them into quantifiable questions and metrics. The selection of the appropriate GQM model and its tailoring needs to be supported. The user will either select an existing model or generate a new one. A new model can be generated from scratch or by reusing pieces of existing models. The degree to which the selection, generation, and reuse tasks can be supported automatically depends largely on the degree to which the GQM paradigm and its templates can be formalized. The user needs to be supported in defining his/her specific goals according to the goal definition template. Based on each goal definition, the TAME system will search for a model in the experience base. If no appropriate model exists, the user will be guided in developing one. Based on the tractability of goals into subgoals and questions the TAME system will identify reusable pieces of existing models and compose as much of an initial model as possible. This initial model will be completed with user interaction. For example, if a user wants to develop a model for assessing a system test method used in a particular environment, the system might compose an initial model by reusing pieces from a model assessing a different test method in the same environment, and from a model for assessing the same system test method in a different environment. A complete GQM model includes rules for interpretation of metrics and guidelines for collecting the prescribed data. The TAME system will automatically generate as much of this information as possible.

(R2) The automatic and manual collection of data and the validation of manually collected data (derived from component C3.2).

The collection of all product-related data (e.g., lines of code, complexity) and certain process-related data (e.g., number of compiler runs, number of test runs) will be completely automated. Automation requires an interface with construction-oriented SEEs. The collection of many process-related data (e.g., effort, changes) and subjective data (e.g., experience of personnel, characteristics of methods used) cannot be automated. The schedule according to which measurement tools are run needs to be defined as part of the planning activity. It is possible to collect data whenever they are needed, periodically (e.g., always at a particular time of the day), or whenever changes of products occur (e.g., whenever a new product version is entered into the experience base all the related metrics are recomputed). All manually collected data need to be validated. Validating whether data are within their defined range, whether all the prescribed data are collected, and whether certain integrity rules among data are maintained will be automated. Some of the measurement tools will be developed as part of the TAME system development project, others will be imported. The need for importing measurement tools will require an effective interconnection mechanism (probably an interconnection language) for integrating tools developed in different languages.

(R3) A mechanism for controlling measurement and analysis (derived from component C3.2).

A GQM model is used to specify and control the execution of a particular analysis and feedback session. According to each GQM model, the TAME system must trigger the execution of measurement tools for data collection, the computation of all metrics and distributions prescribed, and the application of statistical procedures. If certain metrics or distributions cannot be computed due to the lack of data or measurement tools, the TAME system must inform the user.

(R4) A mechanism for interpreting analysis results in a context and providing feedback for the improvement of the execution model, methods and tools (derived from components C3.2, C.5).

We use a GQM model to define the rules and context for interpretation of data and for feedback in order to refine and improve execution models, methods and tools. The degree to which interpretation can be supported depends on our understanding of the software process and product, and the degree to which we express this understanding as formal rules. Today, interpretation rules exist only for some of the aspects of interest and are only valid within a particular project environment or organization. However, interpretation guided by GQM models will enable an evolutionary learning process resulting in better rules for interpretation in the future. The interpretation process can be much more effective provided historical experience is available allowing for the generation of historical baselines. In this case we can at least identify whether observations made during the current project deviate from past experience or not.

(R5) A mechanism for learning in an organization (derived from components C4, C5).

The learning process is supported by iterating the sequence of defining focused goals, refining them into hypotheses, and running experiments. These experiments can range from completely controlled experiments to regular project executions. In each case we apply measurement and analysis procedures to project classes of interest. For each of those classes, a historical experience base needs to be established concerning the effectiveness of the candidate execution models, methods and tools. Feedback from ongoing projects of the same class, the corresponding execution models, methods and tools can be refined and improved with respect to context and precision/detail so that we increase our potential to improve future projects.

The remaining seven (internal) requirements deal with user interface management, report generation, experience base, security and access control, configuration management control, SEE interface and distribution issues. All these issues are important in order to support planning, construction, learning and feedback effectively.

Internal TAME requirements:

(R6) A homogeneous user interface.

We distinguish between the physical and logical user interface. The physical user interface provides a menu or command driven interface between the user and the TAME system. Graphics and window mechansims will be

incorporated whenever useful and possible. The logical user interface reflects the user's view of measurement and analysis. Users will not be allowed to directly access data or run measurement tools. The only way of working with the TAME system is via a GQM model. TAME will enforce this top-down approach to measurement via its logical user interface. The acceptance of this kind of user interface will depend on the effectiveness and ease with which it can be used. Homogeneity is important for both the physical and logical user interface.

(R7) An effective mechanism for presenting data, information, and knowledge.

The presentation of analysis (measurement and interpretation) results via terminal or printer/plotter needs to be supported. Reports need to be generated for different purposes. Project managers will be interested in periodical reports reflecting the current status of their project. High level managers will be interested in reports indicating quality and productivity trends of the organization. The specific interest of each person needs to be defined by one or more GQM models upon which automatic report generation can be based. A laser printer and multicolor plotter would allow the appropriate documentation of tables, histograms, and other kinds of textual and graphical representations.

(R8) The effective storage and retrieval of all relevant data, information, and knowledge in an experience base.

All data, information, and knowledge required to support tailorability and tractability need to be stored in an experience base. Such an experience base needs to store GQM models, engineering products and measurement data. It needs to store data derived from the current project as well as historical data from prior projects. The effectiveness of such an experience base will be improved for the purpose of learning and feedback if, in addition to measurement data, interpretations from various analysis sessions are stored. In the future, the interpretation rules themselves will become integral part of such an experience base. The experience base should be implemented as an abstract data type, accessible through a set of functions and hiding the actual implementation. This latter requirement is especially important due to the fact that current database technology is not suited to properly support software engineering concepts [26]. The implementation of the experience base as an abstract data type allows us to use currently available database technology and substitute more appropriate technology later as it becomes available. The ideal database would be self-adapting to the changing needs of a project environment or an organization. This would require a specification language for software processes and products, and the ability to generate database schemata from specifications written in such a language [46].

(R9) Mechanisms allowing for the implementation of a variety of access control and security strategies.

TAME must control the access of users to the TAME system itself, to various system functions and to the experience base. These are typical functions of a security system. The enforced security strategies depend on the project organization. It is part of planning a project to decide who needs to have access to what functions and pieces of data, information, and knowledge. In addition to these security functions, more sophisticated data access control functions need to be performed. The data access system is expected to "recommend" to a user who is developing a GQM model the kinds of data that might be helpful in answering a particular question and support the process of choosing among similar data based on availability or other criteria.

(R10) Mechanisms allowing for the implementation of a variety of configuration management and control strategies.

In the context of the TAME system we need to manage and control three-dimensional configurations. There is first the traditional product dimension making sure that the various product and document versions are consistent. In addition, each product version needs to be consistent with its related measurement data and the GQM model that guided those measurements. TAME must ensure that a user always knows whether data in the experience base is consistent with the current product version and was collected and interpretated according to a particular model. The actual configuration management and control strategies will result from the project planning activity.

(R11) An interface to a construction-oriented SEE.

An interface between the TAME system (which automates all process model components except for the construction component C3.1 of the TAME process model) and some external SEE (which automates the construction component) is necessary for three reasons: a) to enable the TAME system to collect data (e.g., the number of activations of a compiler, the number of test runs) directly from the actual construction process, b) to enable the TAME system to feed analysis results back into the ongoing construction process, and c) to enable the construction-oriented SEE to store/retrieve products into/from the experience base of the TAME system. Models for appropriate interaction between constructive and analytic processes need to be specified. Interfacing with construction-oriented SEE's poses the problem of efficiently interconnecting systems implemented in different languages and running on different machines (probably with different operating systems).

(R12) A structure suitable for distribution.

TAME will ultimately run on a distributed system consisting of at least one mainframe computer and a number of workstations. The mainframes are required to host the experience base which can be assumed to be very large. The rest of TAME might be replicated on a number of workstations.

B. Architecture

Fig. 2 describes our current view of the TAME architecture in terms of individual architectural components and their control flow interrelationships. The first prototype described in Section IV concentrates on the shaded components of Fig. 2.

We group the TAME components into five logical lev-

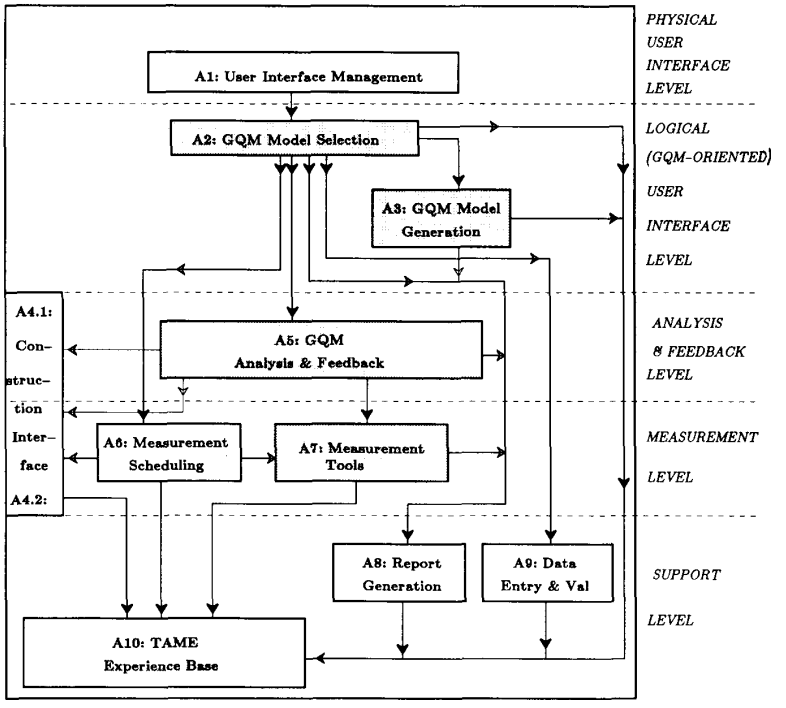

Fig. 2. The architectural design of the TAME system.

els, the physical user interface, logical user interface, analysis and feedback, measurement and support level. Each of these five levels consists of one or more architectural components:

- The Physical User Interface Level consists of one component:

(A1) The User Interface Management component implements the physical user interface requirement R6. It provides a choice of menu or command driven access and supports a window-oriented screen layout.

- The Logical (GQM-Oriented) User Interface Level consists of two components:

(A2) The GQM Model Selection component implements the homogeneity requirement of the logical user interface (R6). It guarantees that no access to the analysis and feedback, measurement, or support level is possible without stating the purpose for access in terms of a specific GQM model.

(A3) The GQM Model Generation component implements requirement R1 regarding the operational and quantifiable definition of GQM models either from scratch or by modifying existing models.

- The Analysis and Feedback Level consists of two components:

(A4.1) This first portion of the Construction Interface component implements the feedback interface between the TAME system and construction-oriented SEEs (part b) of requirement R11).

(A5) The GQM Analysis and Feedback component implements requirement R3 regarding execution and control of an analysis and feedback session, interpretation of the analysis results, and proper feedback. All these activities are done in the context of a GQM model created by A3. The GQM Analysis and Feedback component needs to have access to the specific authorizations of the user in order to know which analysis functions this user can perform. The GQM Analysis and Feedback component also provides analysis functions, for example, telling the user whether certain metrics can be computed based upon the data currently available in the experience base. This analysis feature of the subsystem is used for setting and operationally defining goals, questions, and metrics, as well as actually performing analyses according to those previously established goals, questions, and metrics.

- The Measurement Level consists of three components:

(A4.2) This second portion of the Construction Interface component implements the measurement interface between the TAME system and SEE's (part a) of requirement R11) and the SEE's access to the experience base of the TAME system (part c) of requirement R11).

(A6) The Measurement Scheduling component implements requirement R2 regarding the definition (and execution) of automated data collection strategies. Such strategies for when to collect data via the measurement tools may range from collecting data whenever they are needed for an analysis and feedback session (on-line) to collecting them periodically during low-load times and storing them in the experience base (off-line).

(A7) The Measurement Tools component implements requirement R2 regarding automated data collection. The component needs to be open-ended in order to

allow the inclusion of new and different measurement tools as needed.

- The Support Level consists of three components:

(A8) The Report Generation component implements requirement R7 regarding the production of all kinds of reports.

(A9) The Data Entry and Validation component implements requirement R2 regarding the entering of manually collected data and their validation. Validated data are stored in the experience base component.

(A10) The Experience Base component implements requirement R8 regarding the effective storage and retrieval of all relevant data, information and knowledge. This includes all kinds of products, analytical data (e.g., measurement data, interpretations), and analysis plans (GQM models). This component provides the infrastructure for the operation of all other components of the TAME process model and the necessary interactions among them. The experience base will also provide mechanisms supporting the learning and feedback tasks. These mechanisms include the proper packaging of experience along the context and precision/detail dimensions.

In addition, there exist two orthogonal components which for simplicity reasons are not reflected in Fig. 2:

(A11) The Data Access Control and Security component(s) implement requirement R9. There may exist a number of subcomponents distributed across the logical architectural levels. They will validate user access to the TAME system itself and to various functions at the user interface level. They will also control access to the project experience through both the measurement tools and the experience base.

(A12) The Configuration Management and Control component implements requirement R10. This component can be viewed as part of the interface to the experience base level. Data can only be entered into or retrieved from the experience base under configuration management control.

IV. First TAME Prototye

The first in a series of prototypes is currently being developed for supporting measurement in Ada projects [15]. This first prototype will implement only a subset of the requirements stated in Section III-A because of a) yet unsolved problems that require research, b) solutions that require more formalization, and c) problems with integrating the individual architectural components into a consistent whole. Examples of unsolved problems requiring futher research are the appropriate packaging of the experience along the context and precision/detail dimension and expert system support for interpretation purposes. Examples of solutions requiring more formalization are the GQM templates and the designing of a software engineering experience base. Examples of integration problems are the embedding of feedback loops into the construction process, and the appropriate utilization of data access control and configuration management control mechanisms. At this time, the prototype exists in pieces that have not been fully integrated together as well as partially implemented pieces.

In this section, we discuss for each of the architectural components of this TAME prototype as many of the following issues as are applicable: a) the particular approach chosen for the first prototype, b) experience with this approach, c) the current and planned status of implementation (automation) of the initial approach in the first TAME system prototype, and d) experiences with using the component:

(A1) The User Interface Management component is supposed to provide the physical user interface for accessing all TAME system functions, with the flexibility of choosing between menu and command driven modes and different window layouts. These issues are reasonably well understood by the SEE community. The first TAME prototype implementation will be menu-oriented and based upon the 'X' window mechanism. A primitive version is currently running. This component is currently not very high on our priority list. We expect to import a more sophisticated user interface management component at some later time or leave it completely to parties interested in productizing our prototype system.

(A2) The GQM Model Selection component is supposed to force the TAME user to parameterize each TAME session by first stating the objective of the session in the form of an already existing GQM model or requesting the creation of a new GQM model. The need for this restriction has been derived from the experience that data is frequently misused if it is accessible without a clear goal. The first prototype implementation does not enforce this requirement strictly. The current character of the first prototype as a research vehicle demands more flexibility. There is no question that this component needs to be implemented before the prototype leaves the research environment.

(A3) The GQM Model Generation component is supposed to allow the creation of specific GQM models either from scratch or by modifying existing ones. We have provided a set of templates and guidelines (Section II-B-2). We have been quite successful in the use of the templates and guidelines for defining goals, questions and metrics. There are a large number of organizations and environments in which the model has been applied to specify what data must be collected to evaluate various aspects of the process and product, e.g., NASA/GSFC, Burroughs, AT&T, IBM, Motorola. The application of the GQM paradigm at Hewlett Packard has shown that the templates can be used successfully without our guidance. Several of these experiences have been written up in the literature [4], [16], [17], [39], [48], [56], [60], [61]. We have been less successful in automating the process so that it ties into the experience base. As long as we know the goals and questions *a priori*, the appropriate data can be isolated and collected based upon the GQM paradigm. The first TAME prototype implementation is limited to sup-

port the generation of new models and the modificaton of existing models using an editor enforcing the templates and guidelines. We need to further formalize the templates and guidelines and provide traceability between goals and questions. Formalization of the templates and providing traceability is our most important research issue. In the long run we might consider using artificial intelligence planning techniques.

(A4.1 and A4.2) The Construction Interface component is supposed to support all interactions between a SEE (which supports the construction component of the TAME process model) and the TAME system. The model in Fig. 1 implies that interactions in both directions are required. We have gained experience in manually measuring the construction process by monitoring the execution of a variety of techniques (e.g., code reading [57], testing [20], and CLEANROOM development [61]) in various environments including the SEL [4], [48]. We have also learned how analysis results can be fed back into the ongoing construction process as well as into corporate experience [4], [48]. Architectural component A4.1 is not part of this first TAME prototype. The first prototype implementation of A4.2 is limited to allowing for the integration of (or access to) external product libraries. This minimal interface is needed to have access to the objects for measurement. No interface for the on-line measurement of ongoing construction processes is provided yet.

(A5) The GQM Analysis and Feedback component is supposed to perform analysis according to a specific GQM model. We have gained a lot of experience in evaluating various kinds of experiments and case studies. We have been successful in collecting the appropriate data by tracing GQM models top-down. We have been less successful in providing formal interpretation rules allowing for the bottom-up interpretation of the collected data. One automated approach to providing interpretation and feedback is through expert systems. ARROWSMITH-P provides interpretations of software project data to managers [44]; it has been tested in the SEL/NASA environment. The first prototype TAME implementation triggers the collection of prescribed data (top-down) and presents it to the user for interpretation. The user-provided interpretations will be recorded (via a knowledge acquisition system) in order to accumulate the necessary knowledge that might lead us to identifying interpretation rules in the future.

(A6) The Measurement Scheduling component is supposed to allow the TAME user to define a strategy for actually collecting data by running the measurement tools. Choosing the most appropriate of many possible strategies (requirements Section III-A) might depend on the response times expected from the TAME system or the storage capacity of the experience base. Our experience with this issue is limited because most of our analyses were human scheduled as needed [4], [48]. This component will not be implemented as part of the first prototype. In this prototype, the TAME user will trigger the execution of measurement activities explicitly (which can, of course, be viewed as a minimal implementation supporting a human scheduling strategy).

(A7) The Measurement Tools component is supposed to allow the collection of all kinds of relevant process and product data. We have been successful in generating tools to gather data automatically and have learned from the application of these tools in different environments. Within NASA, for example, we have used a coverage tool to analyze the impact of test plans on the consistency of acceptance test coverage with operational use coverage [53]. We have used a data bindings tool to analyze the structural consistency of implemented systems to their design [41], and studied the relationship between faults and hierarchical structure as measured by the data bindings tool [60]. We have been able to characterize classes of products based upon their syntactic structure [35]. We have not, however, had much experience in automatically collecting process data. The first prototype TAME implementation consists of measurement tools based on the above three. The first tool captures all kinds of basic Ada source code information such as lines of code and structural complexity metrics [35], the second tool computes Ada data binding metrics, and the third tools captures dynamic information such as test coverage metrics [65]. One lesson learned has been that the development of measurement tools for Ada is very often much more than just a reimplementation of similar tools for other languages. This is due to the very different Ada language concepts. Furthermore, we have recognized the importance of having an intermediate representation level allowing for a language independent representation of software product and process aspects. The advantage of such an approach will be that this intermediate representation needs to be generated only once per product or process. All the measurement tools can run on this intermediate representation. This will not only make the actual measurement process less time-consuming but provide a basis for reusing the actual measurement tools to some extent across different language environments. Only the tool generating the intermediate representation needs to be rebuilt for each new implementation language or TAME host enviroment.

(A8) The Report Generator component is supposed to allow the TAME user to produce a variety of reports. The statistics and business communities have commonly accepted approaches for presenting data and interpretations effectively (e.g., histograms). The first TAME prototype implementation does not provide a separate experience base reporting facility. Responsibility for reporting is attached to each individual prototype component; e.g., the GQM Model Generation component provides reports regarding the models, each measurement tool reports on its own measurement data.

(A9) The Data Entry and Validation component is supposed to allow the TAME user to enter all kinds of manually collected data and validate them. Because of the changing needs for measurement, this component must allow for the definition of new (or modification of existing)

data collection forms as well as related validation (integrity) rules. If possible, the experience base should be capable of adapting to new needs based upon new form definitions. We have had lots of experience in designing forms and validations rules, using them, and learning about the complicated issues of deriving validation rules [4], [48]. The first prototype implementation will allow the TAME user to input off-line collected measurement data and validate them based upon a fixed and predefined set of data collection forms [currently in use in NASA's Software Engineering Laboratory (SEL)]. This component is designed but not yet completely implemented. The practical use of the TAME prototype requires that this component provide the flexibility for defining and accepting new form layouts. One research issue is identifying the easiest way to define data collection forms in terms of a grammar that could be used to generate the corresponding screen layout and experience base structure.

(A10) The Experience Base component allows for effective storage and retrieval of all relevant experience ranging from products and process plans (e.g., analysis plans in the form of GQM models) to measurement data and interpretations. The experience base needs to mirror the project environment. Here we are relying on the experience of several faculty members of the database group at the University of Maryland. It has been recognized that current database technology is not sufficient, for several reasons, to truly mirror the needs of software engineering projects [26]. The first prototype TAME implementation is built on top of a relational database management system. A first database schema [46] modeling products as well as measurement data has been implemented. We are currently adding GQM models to the schema. The experiences with this first prototype show that the amount of experience stored and its degree of formalism (mostly data) is not yet sufficient. We need to better package that data in order to create pieces of information or knowledge. The GQM paradigm provides a specification of what data needs to be packaged. However, without more formal interpretation rules, the details of packaging cannot be formalized. In the long run, we might include expert system technology. We have also recognized the need for a number of built-in GQM models that can either be reused without modification or guide the TAME user during the process of creating new GQM models.

(A11) The Data Access Control and Security component is supposed to guarantee that only authorized users can access the TAME system and that each user can only access a predefined window of the experience base. The first prototype implements this component only as far as user access to the entire system is concerned.

(A12) The Configuration Management and Control component is supposed to guarantee consistency between the objects of measurement (products and processes), the plans for measurement (GQM models), the data collected from the objects according to these plans, and the attached interpretations. This component will not be implemented in the first prototype.

The integration of all these architectural components is incomplete. At this point in time we have integrated the first versions of the experience base, three measurement tools, a limited version of the GQM analysis and feedback component, the GQM generation component, and the user interface management component. Many of the UNIX® tools (e.g., editors, print facilities) have been integrated into the first prototype TAME system to compensate for yet missing components. This subset of the first prototype is running on a network of SUN-3's under UNIX. It is implemented in Ada and C.

This first prototype enables the user to generate GQM models using a structured editor. Existing models can be selected by using a unique model name. Support for selecting models based on goal definitions or for reusing existing models for the purpose of generating new models is offered, but the refinement of goals into questions and metrics relies on human intervention. Analysis and feedback sessions can be run according to existing GQM models. Only minimal support for interpretation is provided (e.g., histograms of data). Measurement data are presented to the user according to the underlying model for his/her interpretation. Results can be documented on a line printer. The initial set of measurement tools allows only the computation of a limited number of Ada-source-code-oriented static and dynamic metrics. Similar tools might be used in the case of Fortran source code [33].

V. Summary and Conclusions

We have presented a set of software engineering and measurement principles which we have learned during a dozen years of analyzing software engineering processes and products. These principles have led us to recognize the need for software engineering process models that integrate sound planning and analysis into the construction process.

In order to achieve this integration the software engineering process needs to be tailorable and tractable. We need the ability to tailor the execution process, methods and tools to specific project needs in a way that permits maximum reuse of prior experience. We need to control the process and product because of the flexibility required in performing such a focused development. We also need as much automated support as possible. Thus an integrated software engineering environment needs to support all of these issues.

In the TAME project we have developed an improvement-oriented (integrated) process model. It stresses a) the characterization of the current status of a project environment, b) the planning for improvement integrated into software projects, and c) the execution of the project according to the prescribed project plans. Each of these

®UNIX is a registered trademark of AT&T Bell Laboratories.

tasks must be dealt with from a constructive and analytic perspective.

To integrate the constructive and analytic aspects of software development, we have used the GQM paradigm. It provides a mechanism for formalizing the characterization and planning tasks, controlling and improving projects based on quantitative analysis, learning in a deeper and more systematic way about the software process and product, and feeding back the appropriate experience to current and future projects.

The effectiveness of the TAME process model depends heavily on appropriate automated support by an ISEE. The TAME system is an instantiation of the TAME process model into an ISEE; it is aimed at supporting all aspects of characterization, planning, analysis, learning, and feedback according to the TAME process model. In addition, it formalizes the feedback and learning mechanisms by supporting the synthesis of project experience, the formalization of its representation, and its tailoring towards specific project needs. It does this by supporting goal development into measurement via templates and guidelines, providing analysis of the development and maintenance processes, and creating and using experience bases (ranging from databases of historical data to knowledge bases that incorporate experience from prior projects).

We discussed a limited prototype of the TAME system, which has been developed as the first of a series of prototypes that will be built using an iterative enhancement model. The limitations of this prototype fall into two categories, limitations of the technology and the need to better formalize the model so that it can be automated.

The short range (1–3 years) goal for the TAME system is to build the analysis environment. The mid-range goal (3–5 years) is to integrate the system into one or more existing or future development or maintenance environments. The long range goal (5–8 years) is to tailor those environments for specific organizations and projects.

The TAME project is ambitious. It is assumed it will evolve over time and that we will learn a great deal from formalizing the various aspects of the TAME project as well as integrating the various paradigms. Research is needed in many areas before the idealized TAME system can be built. Major areas of study include measurement, databases, artificial intelligence, and systems. Specific activities needed to support TAME include: more formalization of the GQM paradigm, the definition of better models for various quality and productivity aspects, mechanisms for better formalizing the reuse and tailoring of project experience, the interpretation of metrics with respect to goals, interconnection languages, language independent representation of software, access control in general and security in particular, software engineering database definition, configuration management and control, and distributed system architecture. We are interested in the role of further researching the ideas and principles of the TAME project. We will build a series of evolving prototypes of the system in order to learn and test out ideas.

ACKNOWLEDGMENT

The authors thank all their students for many helpful suggestions. We especially acknowledge the many contributions to the TAME project and, thereby indirectly to this paper, by J. Bailey, C. Brophy, M. Daskalantonakis, A. Delis, D. Doubleday, F. Y. Farhat, R. Jeffery, E. E. Katz, A. Kouchakdjian, L. Mark, K. Reed, Y. Rong, T. Sunazuka, P. D. Stotts, B. Swain, A. J. Turner, B. Ulery, S. Wang, and L. Wu. We thank the guest editors and external reviewers for their constructive comments.

REFERENCES

[1] W. Agresti, "SEL Ada experiment: Status and design experience," in *Proc. Eleventh Annu. Software Engineering Workshop*, NASA Goddard Space Flight Center, Greenbelt, MD, Dec. 1986.

[2] J. Bailey and V. R. Basili, "A meta-model for software development resource expenditures," in *Proc. Fifth Int. Conf. Software Engineering*, San Diego, CA, Mar. 1981, pp. 107–116.

[3] V. R. Basili, "Quantitative evaluation of software engineering methodology," in *Proc. First Pan Pacific Computer Conf.*, Melbourne, Australia, Sept. 1985; also available as Tech. Rep. TR-1519, Dep. Comput. Sci., Univ. Maryland, College Park, July 1985.

[4] V. R. Basili, "Can we measure software technology: Lessons learned from 8 years of trying," in *Proc. Tenth Annu. Software Engineering Workshop*, NASA Goddard Space Flight Center, Greenbelt, MD, Dec. 1985.

[5] ——, "Evaluating software characteristics: Assessement of software measures in the Software Engineering Laboratory," in *Proc. Sixth Annu. Software Engineering Workshop*, NASA Goddard Space Flight Center, Greenbelt, MD, 1981.

[6] V. R. Basili and J. Beane, "Can the Parr curve help with the manpower distribution and resource estimation problems," *J. Syst. Software*, vol. 2, no. 1, pp. 59–69, 1981.

[7] V. R. Basili and K. Freburger, "Programming measurement and estimation in the Software Engineering Laboratory," *J. Syst. Software*, vol. 2, no. 1, pp. 47–57, 1981.

[8] V. R. Basili and D. H. Hutchens, "An empirical study of a syntactic measure family," *IEEE Trans. Software Eng.*, vol. SE-9, no. 11, pp. 664–672, Nov. 1983.

[9] V. R. Basili and E. E. Katz, "Measures of interest in an Ada development," in *Proc. IEEE Comput. Soc. Workshop Software Engineering Technology Transfer*, Miami, FL, Apr. 1983, pp. 22–29.

[10] V. R. Basili, E. E. Katz, N. M. Panlilio-Yap, C. Loggia Ramsey, and S. Chang, "Characterization of an Ada software development," *Computer*, pp. 53–65, Sept. 1985.

[11] V. R. Basili and C. Loggia Ramsey, "ARROWSMITH-P: A prototype expert system for software engineering management," in *Proc. IEEE Symp. Expert Systems in Government*, Oct. 23–25, 1985, pp. 252–264.

[12] V. R. Basili and N. M. Panlilio-Yap, "Finding relationships between effort and other variables in the SEL," in *Proc. IEEE COMPSAC*, Oct. 1985.

[13] V. R. Basili and B. Perricone, "Software errrors and complexity: An empirical investigation," *ACM, Commun.*, vol. 27, no. 1, pp. 45–52, Jan. 1984.

[14] V. R. Basili and R. Reiter, Jr., "A controlled experiment quantitatively comparing software development approaches," *IEEE Trans. Software Eng.*, vol. SE-7, no. 5, pp. 299–320, May 1981.

[15] V. R. Basili and H. D. Rombach, "TAME: Tailoring an Ada measurement environment," in *Proc. Joint Ada Conf.*, Arlington, VA, Mar. 16–19, 1987, pp. 318–325.

[16] ——, "Tailoring the software process to project goals and environments," in *Proc. Ninth Int. Conf. Software Engineering*, Monterey, CA, Mar. 30–Apr. 2, 1987, pp. 345–357.

[17] ——, "TAME: Integrating measurement into software environments," Dep. Comput. Sci., Univ. Maryland, College Park, Tech. Rep. TR-1764 (TAME-TR-1-1987), June 1987.

[18] ——, "Software reuse: A framework," in *Proc. Tenth Minnowbrook Workshop Software Reuse*, Blue Mountain Lake, NY, Aug. 1987.
[19] V. R. Basili and R. W. Selby, Jr., "Data collection and analysis in software research and management," in *Proc. Amer. Statist. Ass. and Biomeasure Soc. Joint Statistical Meetings*, Philadelphia, PA, Aug. 13-16, 1984.
[20] ——, "Comparing the effectiveness of software testing strategies," *IEEE Trans. Software Eng.*, vol. SE-13, no. 12, pp. 1278-1296, Dec. 1987.
[21] ——, "Calculation and use of an environment's characteristic software metric set," in *Proc. Eighth Int. Conf. Software Engineering*, London, England, Aug. 1985.
[22] V. R. Basili, R. W. Selby, and D. H. Hutchens, "Experimentation in software engineering," *IEEE Trans. Software Eng.*, vol. SE-12, no. 7, pp. 733-743, July 1986.
[23] V. R. Basili, R. W. Selby, and T.-Y. Phillips, "Metric analysis and data validation across Fortran projects," *IEEE Trans. Software Eng.*, vol. SE-9, no. 6, pp. 652-663, Nov. 1983.
[24] V. R. Basili and A. J. Turner, "Iterative enhancement: A practical technique for software development," *IEEE Trans. Software Eng.*, vol. SE-1, no. 4, pp. 390-396, Dec. 1975.
[25] V. R. Basili and D. M. Weiss, "A methodology for collecting valid software engineering data," *IEEE Trans. Software Eng.*, vol. SE-10, no. 3, pp. 728-738, Nov. 1984.
[26] P. A. Bernstein, "Database system support for software engineering," in *Proc. Ninth Int. Conf. Software Engineering*, Monterey, CA, Mar. 30-Apr. 2, 1987, pp. 166-178.
[27] D. Bjorner, "On the use of formal methods in software development," in *Proc. Ninth Int. Conf. Software Engineering*, Monterey, CA, Mar. 30-Apr. 2, 1987, pp. 17-29.
[28] B. W. Boehm, "Software engineering," *IEEE Trans. Comput.*, vol. C-25, no. 12, pp. 1226-1241, Dec. 1976.
[29] ——, *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, 1981.
[30] ——, "A spiral model of software development and enhancement," *ACM Software Eng. Notes*, vol. 11, no. 4, pp. 22-42, Aug. 1986.
[31] B. W. Boehm, J. R. Brown, and M. Lipow, "Quantitative evaluation of software quality," in *Proc. Second Int. Conf. Software Engineering*, 1976, pp. 592-605.
[32] C. Brophy, W. Agresti, and V. R. Basili, "Lessons learned in use of Ada oriented design methods," in *Proc. Joint Ada Conf.*, Arlington, VA, Mar. 16-19, 1987, pp. 231-236.
[33] W. J. Decker and W. A. Taylor, "Fortran static source code analyzer program (SAP)," NASA Goddard Space Flight Center, Greenbelt, MD, Tech. Rep. SEL-82-002, Aug. 1982.
[34] C. W. Doerflinger and V. R. Basili, "Monitoring software development through dynamic variables," *IEEE Trans. Software Eng.*, vol. SE-11, no. 9, pp. 978-985, Sept. 1985.
[35] D. L. Doubleday, "ASAP: An Ada static source code analyzer program," Dep. Comput. Sci., Univ. Maryland, College Park, Tech. Rep. TR-1895, Aug. 1987.
[36] M. Dowson, "ISTAR—An integrated project support environment," in *ACM Sigplan Notices (Proc. Second ACM Software Eng. Symp. Practical Development Support Environments)*, vol. 2, no. 1, Jan. 1987.
[37] M. Dyer, "Cleanroom software development method," IBM Federal Systems Division, Bethesda, MD, Oct. 14, 1982.
[38] J. Gannon, E. E. Katz, and V. R. Basili, "Measures for Ada packages: An initial study," *Commun. ACM*, vol. 29, no. 7, pp. 616-623, July 1986.
[39] R. B. Grady, "Measuring and managing software maintenance," *IEEE Software*, vol. 4, no. 5, pp. 35-45, Sept. 1987.
[40] M. H. Halstead, *Elements of Software Science*. New York: Elsevier North-Holland, 1977.
[41] D. H. Hutchens and V. R. Basili, "System structure analysis: Clustering with data bindings," *IEEE Trans. Software Eng.*, vol. SE-11, pp. 749-757, Aug. 1985.
[42] E. E. Katz and V. R. Basili, "Examining the modularity of Ada programs," in *Proc. Joint Ada Conf.*, Arlington, VA, Mar. 16-19, 1987, pp. 390-396.
[43] E. E. Katz, H. D. Rombach, and V. R. Basili, "Structure and maintainability of Ada programs: Can we measure the differences?" in *Proc. Ninth Minnowbrook Workshop Software Performance Evaluation*, Blue Mountain Lake, NY, Aug. 5-8, 1986.
[44] C. Loggia Ramsey and V. R. Basili, "An evaluation of expert systems for software engineering management," Dep. Comput. Sci., Univ. Maryland, College Park, Tech. Rep. TR-1708, Sept. 1986.
[45] M. Marcus, K. Sattley, S. C. Schaffner, and E. Albert, "DAPSE: A distributed Ada programming support environment," in *Proc. IEEE Second Int. Conf. Ada Applications and Environments*, 1986, pp. 115-125.
[46] L. Mark and H. D. Rombach, "A meta information base for software engineering," Dep. Comput. Sci., Univ. Maryland, College Park, Tech. Rep. TR-1765, July 1987.
[47] T. J. McCabe, "A complexity measure," *IEEE Trans. Software Eng.*, vol. SE-2, no. 4, pp. 308-320, Dec. 1976.
[48] F. E. McGarry, "Recent SEL studies," in *Proc. Tenth Annu. Software Engineering Workshop*, NASA Goddard Space Flight Center, Greenbelt, MD, Dec. 1985.
[49] L. Osterweil, "Software processes are software too," in *Proc. Ninth Int. Conf. Software Engineering*, Monterey, CA, Mar. 30-Apr. 2, 1987, pp. 2-13.
[50] F. N. Parr, "An alternative to the Rayleigh curve model for software development effort," *IEEE Trans. Software Eng.*, vol. SE-6, no. 5, pp. 291-296, May 1980.
[51] L. Putnam, "A general empirical solution to the macro software sizing and estimating problem," *IEEE Trans. Software Eng.*, vol. SE-4, no. 4, pp. 345-361, Apr. 1978.
[52] C. V. Ramamoorthy, Y. Usuda, W.-T. Tsai, and A. Prakash, "GENESIS: An integrated environment for supporting development and evolution of software," in *Proc. COMPSAC*, 1985.
[53] J. Ramsey and V. R. Basili, "Analyzing the test process using structural coverage," in *Proc. Eighth Int. Conf. Software Engineering*, London, England, Aug. 1985, pp. 306-311.
[54] H. D. Rombach, "Software design metrics for maintenance," in *Proc. Ninth Annu. Software Engineering Workshop*, NASA Goddard Space Flight Center, Greenbelt, MD, Nov. 1984.
[55] ——, "A controlled experiment on the impact of software structure on maintainability," *IEEE Trans. Software Eng.*, vol. SE-13, no. 3, pp. 344-354, Mar. 1987.
[56] H. D. Rombach and V. R. Basili, "A quantitative assessment of software maintenance: An industrial case study," in *Proc. Conf. Software Maintenance*, Austin, TX, Sept. 1987, pp. 134-144.
[57] H. D. Rombach, V. R. Basili, and R. W. Selby, Jr., "The role of code reading in the software life cycle," in *Proc. Ninth Minnowbrook Workshop Software Performance Evaluation*, Blue Mountain Lake, NY, August 5-8, 1986.
[58] W. W. Royce, "Managing the development of large software systems: Concepts and techniques," in *Proc. WESCON*, Aug. 1970.
[59] R. W. Selby, Jr., "Incorporating metrics into a software environment," in *Proc. Joint Ada Conf.*, Arlington, VA, Mar. 16-19, 1987, pp. 326-333.
[60] R. W. Selby and V. R. Basili, "Analyzing error-prone system coupling and cohesion," Dep. Comput. Sci., Univ. Maryland, College Park, Tech. Rep., in preparation.
[61] R. W. Selby, Jr., V. R. Basili, and T. Baker, "CLEANROOM software development: An empirical evaluation," *IEEE Trans. Software Eng.*, vol. SE-13, no. 9, pp. 1027-1037, Sept. 1987.
[62] C. E. Walston and C. P. Felix, "A method of programming measurement and estimation," *IBM Syst. J.*, vol. 16, no. 1, pp. 54-73, 1977.
[63] A. I. Wasserman and P. A. Pircher, "Visible connections," *UNIX Rev.*, Oct. 1986.
[64] *Webster's New Collegiate Dictionary*. Springfield, MA: Merriam, 1981.
[65] L. Wu, V. R. Basili, and K. Reed, "A structure coverage tool for Ada software systems," in *Proc. Joint Ada Conf.*, Arlington, VA, Mar. 16-19, 1987, pp. 294-303.
[66] M. Zelkowitz, R. Yeh, R. Hamlet, J. Gannon, and V. R. Basili, "Software engineering practices in the U.S. and Japan," *Computer*, pp. 57-66, June 1984.

Victor R. Basili (M'83-SM'84) is Professor and Chairman of the Department of Computer Science at the University of Maryland, College Park. He was involved in the design and development of several software projects, including the SIMPL family of programming languages. He is currently measuring and evaluating software development in industrial and government settings and has consulted with many agencies and organizations, including IBM, GE, CSC, GTE, MCC, AT&T, Motorola, HP, NRL, NSWC, and NASA. He is

one of the founders and principals in the Software Engineering Laboratory, a joint venture between NASA Goddard Space Flight Center, the University of Maryland and Computer Sciences Corporation, established in 1976. He has been working on the development of quantitative approaches for software management, engineering, and quality assurance by developing models and metrics for the software development process and product. He has authored over 90 papers. In 1982, he received the Outstanding Paper Award from the IEEE TRANSACTIONS ON SOFTWARE ENGINEERING for his paper on the evaluation of methodologies.

Dr. Basili is currently the Editor-in-Chief of the IEEE TRANSACTIONS ON SOFTWARE ENGINEERING and was Program Chairman for several conferences including the 6th International Conference on Software Engineering. He has served on the Editorial Board of the *Journal of Systems and Software*. He is a member of the Board of Governors of the IEEE Computer Society.

H. Dieter Rombach received the B.S. degree (Vordiplom) in mathematics and the M.S. degree (Diplom) in mathematics and computer science from the University of Karlsruhe, West Germany, and the Ph.D. degree (Dr. rer. nat.) in computer science from the University of Kaiserslautern, West Germany.

He is an Assistant Professor of Computer Science at the University of Maryland, College Park. He is also affiliated with the University of Maryland Institute for Advanced Computer Studies (UMIACS) and the Software Engineering Laboratory (SEL), a joint venture between NASA Goddard Space Flight Center, the University of Maryland, and Computer Sciences Corporation. His research interests include software methodologies, measurement of the software process and its products, software engineering environments, and distributed systems.

Dr. Rombach served as Guest Editor for the *IEEE Software* magazine Special Issue on Software Quality Assurance (September 1987). He is a member of the IEEE Computer Society, the Association for Computing Machinery, and the German Computer Society (GI).