Technical Report TR-1764
TAME-TR-1-1987

June 1987

T A M E: INTEGRATING MEASUREMENT INTO SOFTWARE ENVIRONMENTS

Victor R. Basili and H. Dieter Rombach

Department of Computer Science
University of Maryland
College Park MD 20742
(301) 454-2002 or -8974

* Research for this study was supported in part by NASA grant NSG-5123 and ONR 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.

T A M E:
INTEGRATING MEASUREMENT INTO SW ENVIRONMENTS

Victor R. Basili and H. Dieter Rombach

Department of Computer Science
University of Maryland
College Park MD 20742
(301) 454–2002 or –8974

Abstract

Based upon a dozen years of analyzing software engineering processes and products, we propose a set of software engineering process and measurement principles. These principles lead to the view that an Integrated Software Engineering Environment (ISEE) should support multiple process models across the full software life cycle, the technical and management aspects of software engineering, and the planning, construction, and feedback and learning activities. These activities need to be tailored to the specific project under development and they must be tractable for management control. The tailorability and tractability attributes require the support of a measurement process. The measurement process needs to be top–down, based upon operationally defined goals.

The TAME project uses the goal/question/metric paradigm to support this type of measurement paradigm. It provides for the establishment of project specific goals and corporate goals for planning, provides for the tracing of these goals throughout the software life cycle via feedback and post mortem analysis, and offers a mechanism for long range improvement of all aspects of software development.

The TAME system automates as much of this process as possible, by supporting goal development into measurement via models and templates, providing evaluation and analysis of the development and maintenance processes, and creating and using databases of historical data and knowledge bases that incorporate experience from prior projects.

* Research for this study was supported in part by NASA grant NSG–5123 and ONR 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 Mary-land.

TABLE OF CONTENTS:

1. Introduction

The **software engineering process** needs 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 [55]. The software process requires tailorability because the overall process 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 [55]. The software process requires tractability because it needs to be planned, the various planned steps in 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.

The goal of a **software engineering environment (SEE)** should be to support a tailorable and tractable software engineering process by automating as much of it as possible. Currently, SEEs are often designed based upon the experience of the building organization and the state-of-the-art knowledge of what can be automated. The concentration is mostly on constructive tools, i.e. tools that help in the building of a software product, rather than analytic tools, i.e. tools that help in the planning, understanding, learning and feedback process.

At the University of Maryland, we have been working on providing such planning, understanding, learning and feedback support for SEEs. The **TAME (Tailoring A Measurement Environment) project** is based on the idea that the software engineering process needs to be tailored and tracked for the project specific quality and productivity goals, the characteristics of the project environment, and the overall organization. Therefore, management and engineering need a mechanism to help them define their goals operationally, tailor them to the project needs, and evaluate their success and failure in achieving these goals. The mechanism we use to achieve this goal is **measurement**. The measurement process must be top-down, i.e. driven by operationally defined goals, permit a bottom-up interpretation within the appropriate context, and provide feedback for improving the processes and the products from the perspective of the specific project and the overall organization.

It seems appropriate to define **some of the important terms** used in this paper in a intuitive form. The term *engineering* comprises both development and maintenance. A software engineering*project* is embedded in some *project environment* (for example, personnel, type of application) and within some *organization* (e.g. NASA, IBM). The software project is conducted according to a software engineering *process* which is defined in terms of an *overall process model* (e.g. waterfall model [28, 51], iterative enhancement model [24], spiral model [30]) and supplementing *techniques (methods, tools)*. We distinguish between *constructive* and *analytic* methods and tools. Whereas constructive methods and tools are concerned with building products, analytic methods and tools are concerned with analyzing the constructive process and the resulting products. The project personnel is categorized as either *engineers* (e.g., designers, coders, testers) or *managers*.

In the first part of this paper we list empirically derived principles about the software engineering process in general (section 2) and the measurement process in particular (section 3), and derive an abstract software engineering model based upon these principles (section 4). In the second part of this paper we describe characteristics of SEEs to properly support this particular software engineering model (section 5) and introduce the TAME system, an automated measurement and evaluation environment, designed to support the suggested approach to measurement in an SEE (section 6). Finally we describe the first TAME prototype, which is the first of a series of prototypes being built using an iterative enhancement model (section 7).

2. Software Process Principles

Based upon our experience in monitoring and evaluating the software engineering processes and products in a variety of organizations over the past dozen years [4, 42], we have learned several lessons about the software engineering process that are phrased as principles for the TAME project:

The first five principles deal with the tractability of software engineering processes and products, i.e. how should we plan for quality software engineering, how should we evaluate the actual quality of performed processes and created products, and how can we learn from this evaluation process.

(P1) It is necessary to develop quality a priori. Quality cannot be tested into the system or even inspected into the system. We must improve the construction process and not rely on analysis techniques (e.g. quality assurance techniques) as a substitute for construction techniques.

We need to spend more time developing better methods for constructing (e.g. specifying, designing, coding) software. Even though some of the formalisms developed are still not easy to use on the construction of large systems, several notable successes have begun to emerge, even when the formalisms are partially or informally applied [27, 36].

Too often, methods that were meant for quality analysis, e.g. testing have become methods for construction. This means that we debug code or design rather than use the test process as a quality assurance method. Process models like the one underlying the Cleanroom approach [36] are based upon the idea that testing can be used only as a quality assurance technique with a great deal of improvement in the quality of the system. We have run replicated experiments with this approach and found that it truly adds to the quality of the product [53].

(P2) In order to develop quality a priori, we need to formalize the planning of the software engineering process.

Our frequent inability to develop quality a priori is due to a lack of planning the constructive processes as well as analytic processes. Without such plans the trial and error approach can hardly be avoided. Proper planning of software engineering processes requires knowledge concerning the impact of techniques, given the characteristics of a particular project. We need to provide better formalism for planning as well as more knowledge concerning the effectiveness of techniques. The goal/question/metric paradigm [3, 19, 20, 25] is an attempt towards formalizing the planning of analytic processes.

(P3) Engineering methods are often heuristic and not formal. They require interpretation and evaluation as to whether they are being performed appropriately, if at all.

Our experience in trying to characterize methods, so that they can be evaluated as to whether they are being applied correctly, demonstrates the lack of precision in the specification of the methods [12, 43]. In a study of the state of the practice in industry, we found that very few organizations are using the methods and tools appropriately [57]. This is largely due to the heuristic nature of many of the methods, i.e. if one engineer "does it right" it is hard to explain to another engineer exactly what he/she is doing. Other reasons for improper use of methods and tools are the lack of training, or inappropriateness of the methods and tools for the problem and experience of the engineer or programmer.

(P4) In order to improve software engineering in an organization, we need to formalize the evaluation and improvement of software engineering processes and products.

Based upon the poor performance in the use of methods and tools, there is a need to evaluate not only the product but the processes used in software engineering. There needs to be a mechanism for evaluating how well the process is being performed so that it can be improved. An evaluation and improvement paradigm is presented in [3, 25]. In our study [57] we found very few organizations running post mortem analyses that would allow them to learn how to better construct software.

(P5) Software engineers and managers need feedback in real time and the organization needs post mortem analysis in order to improve the process and product.

We found that many project managers keep track of some data during development and maintenance but because this information is often manual and the data is inconsistent and incomplete, it tends to be useless for real feedback and post mortem analysis [57].

The rest of these principles deal with the tailorability of the software engineering processes and products. The first three principles stress the fact that all software engineering environments are different, that there is a need for tailoring the specific process model, methods and tools to the particular project environment, and that this tailoring process needs to be formalized. The next principle emphasizes that this need for tailoring does not exclude extensive reuse of experience. The final principle deals with the requirements imposed by these kinds of tailored projects in terms of management flexibility and automated support.

(P6) All project environments and products are different in some way. These differences must be taken into account in the software engineering processes and in the quality goals set for the products.

Almost every organization we looked at had a software engineering manual but the managers of projects of that organization almost never used it. There explanation was that their project did not match the "ideal" project description for which the guidebook was written [57].

(P7) There are many process models for software engineering. The process model needs to be tailored to the organization and project needs and characteristics.

It was clear that each project needed to be able to tailor the process specified in the guidebook, but here was no mechanism to help with this tailoring process. For example, for a project in which the product is an application that the engineering group has built before, the process and the various methods and tools may be quite different than for a project which involves a new application for the organization [15].

(P8) We need to formalize the tailoring of processes towards the quality and productivity goals of the project and the characteristics of the project environment and the organization.

Our only hope is to support the process by helping the engineers and managers establish goals for projects, tailor the methods and tools for those goals, evaluate whether or not those goals have been achieved, and learn from what they have done so the next project can be performed better [19].

(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.

Organizations like NASA have a great deal of experience in reusing processes, and experience [2, 6, 8]. This is brought out significantly by the fact that in trying new methods the project's productivity drops significantly [1, 32]. Unfortunately, most of the reuse experience lies in the people, rather than in the institution.

(P10) Because of this, management control is crucial and must be flexible. Management must be supported in this process.

It is difficult for managers to keep track of all the factors and experience in their heads. They must make decisions based upon local project factors. There must be an automated support mechanism that helps in the process.

3. Software Measurement Principles

The need for tailorability and tractability as attributes of a software engineering process designed to support these process principles is obvious. Tailorability is dependent upon our understanding of the project goals, the characteristics of the project environment and the organization, and the effects of a set of candidate process models, methods and tools in similar project environments for similar goals. For example, in order to tailor the design inspection process for a particular project, we need to know the level of reliability required for the product, the distribution of errors likely to be made due to the experience of our developers, and the effects of different variations of the design inspection process on the critical error types. Tractability is also dependent on our understanding of the project goals and the characteristics of the project environment and the organization, but also on our ability to specify and perform those steps that are important to the project goals. For example, in order to track the design inspection process with respect to a desired level of reliability, we need to be able to specify, in an understandable way, each step in the design inspection process and its effect on the reliability of the final product based upon such things as the experience of our developers and inspectors, and evaluate its execution relative to that specification.

The ability to tailor and track the software engineering process requires a **comprehensive analytic approach** to understanding. A **top down measurement process**, i.e. defining a set of goals in an operational way that lead to metrics, supports such a comprehensive analytical approach. For example, the proper evaluation of the effectiveness of the inspection process requires measuring the expertise of inspectors on a relative scale, the number and types of detected errors, and the relationship between faults found during inspection and reliability.

Again, based upon over a dozen years in measuring the software development process and product in a variety of environments, we have recognized the following principles (including the top-down orientation) for performing measurement:

The first four principles deal with 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 motivating the various aspects of software engineering processes and products.

We need to characterize in order to distinguish the factors that differentiate the software processes and products, and provide an historical database for future comparisons. We cannot evaluate without comparing and we cannot compare unless we are sure we are comparing similar things. Evaluation is especially important for tailoring (e.g. which tools work best in what cases). Prediction and motivation are needed for planning.

(M2) Measurement must be taken on both the software processes and the various software products.

There has been a fair amount of work on measuring the software product, even though most of this work has focused on the final product and the code in particular [29, 38, 41 etc.]. Monitoring the process and all types of products is important if we are to assess the quality of the products delivered. But there has not been enough work done on measuring the other products of the software development, e.g. the requirements document, the design documents [47], or the test plans and the relationship between these products. These products and their relationship can be measured and the analysis can provide insight into the environment and the project for evaluation and improvement [46].

However, it is very important that we assess the software processes used in developing these products. This is important for planning , constructing, and learning if we believe there is a relationship between the quality of the process and the quality of the product. We need to evaluate whether the process has been performed correctly in order to evaluate it. There is evidence that this relationship exists and can be measured during development [5, 17, 53] as well as during maintenance [48, 49]. There is also evidence that these techniques are not being performed correctly in many environment [57].

(M3) There are a variety of uses for measurement. The purpose of the measurement should be clearly stated. We can use measurement to examine cost, effectiveness, reliability, correctness, maintainability, efficiency, user friendliness, etc.

We have used measurement for many of these purposes [7, 9, 10, 11, 15, 16, 17, 19, 21, 23, 25, 50, 53]. However we have found that if the purposes for the measurement are not clearly articulated, it is difficult to organize the appropriate data needed and interpret it appropriately [57].

(M4) Measurement needs to be viewed from the appropriate perspective. The corporation, the manager, the developer, the customer's organization and the user, all view the product and the process from a different perspective and thus may want to know different things about the project and to different levels of detail.

Most measurement in industry is collected from the point of view of the individual project manager [57]. Thus, for example, the corporation can't make use of the data because there is little commonality in the definitions and goals of different managers.

The rest of these principles deal with how the measurement process must be performed. The first two discuss characteristics of metrics (i.e. what kinds of metrics, how many are needed), while the rest deal with the characteristics of the measurement process (i.e. what should the measurement process look like, how do we support planning, construction, and learning and feedback).

(M5) Subjective as well as objective metrics are required. We are not yet able to objectively analyze all kinds of information but there is a great deal of knowledge is needed and available that can be obtained by a careful characterization of subjective knowledge.

For example as stated earlier, if we are to evaluate the effectiveness of various processes, such as design inspections, then we need to know how well the developers understood the technique and the application/problem, and how well they applied the technique before the effectiveness of the technique can be assessed. This type of measurement can only be performed by subjective measurement at present. Subjective evaluation can be categorized on a quantitative scale to a reasonable degree of accuracy [5, 19]. We have applied such subjective characterization schemes in several environments with satisfactory results (e.g. the Software Engineering Laboratory [4, 42], IBM, AT&T, Burroughs [49]).

(M6) Metrics in isolation are useless. For both definition and interpretation purposes, a set of metrics need to be defined that frame the purpose for the measurement process. We call this set a metric vector.

We have been able to show that a set of metrics can be used to provide insight into the characteristics of the product and the process [24]. Different vectors may be associated with the various levels of the product hierarchy, e.g. the full product, the various subsystems, the modules, etc. In each case, metrics such as source lines of code, inter–data complexity, intra–data complexity, etc can be associated with a particular part of the system. For example, a metric like lines of source code only provides a very small insight into the size of a product. We need other metrics like number of executable statements and number of lines of comments to help understand the concept of size.

(M7) The development and maintenance environments must be prepared for measurement and evaluation. There is a planning phase necessary for this activity and the activity must be carefully embedded in the process. This planning phase must take into account the experimental design appropriate for the situation.

It is necessary to decide what we want to measure, how we are going to measure it and how we are going to interpret the results. Part of the planning process deals with choosing the appropriate set of metrics, not too many or too few, evaluating the cost of collection and analysis, and determining how it will be used. Often data is collected but not used because it it was not appropriately planned for [57]. Part of the problem is in laying out the experimental scheme appropriate for the kinds of assessment required. We have identified four such experimental layouts for varying degrees of cost vs. certitude in the experimental results [20].

(M8) We cannot just use other peoples models and metrics as defined. They must be tailored for the environment in which they are applied and checked for validity in that environment.

We have tried to apply a variety of existing models, for such things as resource allocation [31, 44, 45, 54] in different environments and found that they did not work as well as they did in the environment for which they were developed, if at all [2, 6, 8, 14]. This is because there are some many factors involved in software development and each environment is different. We have tried to apply various metrics [38, 41, etc] in various environment and have found very mixed results [9, 23].

(M9) In order to define a set of operational goals, specify the appropriate metrics, permit valid contextual interpretation and evaluation, and provide feedback for tailorability and tractability. The measurement process must be top–down rather than bottom up. We use the goal/question/metric paradigm for this purpose.

The problem is that measurement must be associated with the specific environment and taken for a specific set of purposes. The question of what to measure depends upon what it is you want to know. Therefore it is important that the metrics represent answers to specific questions and goals set for the software development process and product. One reason why metrics are collected but not used [57] is because the measurement process was not organized correctly and the metrics were difficult to interpret because they were not defined based upon a set of operational goals. To aid in this process, we developed the goal/question metric paradigm [25]. It supports the development of goals which can be refined into questions which motivate the metrics, as well as providing a context for interpretation and analysis of the metrics [3].

(M10) There is a subset of measures that provides the needed information for definition and interpretation purposes. We call this a characteristic set of metrics for the local environment.

We have been able to show that a limited set of metrics can be used to provide insight into the characteristics of the product and the process [22]. This characteristic metric set can define the minimum necessary information for characterization, evaluation, prediction and motivation.

(M11) Data can be collected via forms, interviews, and automatically via analyzers of the various products. Data collected via forms and interviews requires validation.

We have used all of the mechanisms for data collection in the SEL [4, 42]. We have found that there is a need to validate the data collected via forms, such as error and change data, [25]. The validation process requires a data analyst with an understanding of the data but could be simplified by automated support.

(M12) In order to evaluate and compare projects and to develop models we need an historical database. This database should characterize the local environment.

This database can act as a basis for comparison for new projects. For example, we have been able to show such things as common fault and error histories in certain environments [25] and the differences represented by new developments [15].

(M13) Metrics must be associated with interpretations, but these interpretations must be given in context.

This database can be supplied with interpretations for various values of sets of metrics and used to assess process and product characteristics. The database provides a standard value range for various metrics. When these values deviate from the norm there is a sign that there is something different about this project. The interpretations associated with the project can provide insights to management as to what is different, i.e. whether the project is in better or worse shape than the normal project. This technique was used in the SEL to provide NASA management with project assessment on an experimental basis and proved effective [34].

(M14) This database should evolve to formalize expert knowledge.

The database and metrics with associated interpretations can be combined into a knowledge base that will help track project progress and advise managers. A prototype expert system, Arrowsmith-P was built to demonstrate the feasibility of an expert system for software management built upon this concept [13]. Experiments with this knowledge base system proved to be as good as the expert managers in predicting problems with software project [46]. Although this system and the experiment were based on a limited set of metrics and performed in a homogeneous software environment (i.e. similar systems built under similar circumstances), it is believed that since the knowledge base can track more data and more projects than any manager, a system can be built that does better than most managers.

4. Software Engineering Process

The software process principles as well as the measurement principles (aiming at supporting the process attributes tailorability and tractability) suggest that sound software engineering (at a very abstract level) needs to be concerned with **planning**, **construction**, and **learning & feedback**.

- **Planning** the software engineering process is aimed at providing a basis for developing quality a priori (principle P1). It includes choosing the appropriate overall process model as well as the specific methods and tools (principles P6 and P7). It involves tailoring each of them for the project specific goals and the characteristics of the project environment and the organization. The constructive and the analytic process models, methods and tools need to be planned. The effectiveness of this planning process depends on the precision in the specification of engineering processes and methods (formal is better than heuristic: principle P3) and the experience concerning their effect (only to be reused after tailoring: principle P9). This kind of experience can be gained via measurement and made accessible through historical databases (principle M12) or even expert systems (principle M14). The entire planning process (principle P2), the tailoring process (principle P8), as well as the measurement process need to be formalized. The formalization of measurement needs to include deriving the appropriate metrics (principle M10) for defined evaluation goals (principle M9), experimental design (principle M7) and data collection and validation (principle M11).

  The **goal/question/metric** paradigm was developed as a mechanism for formalizing this kind of measurement [3, 19, 20, 25]. It represents a systematic approach for determining the goal of measurement (tailored to the specific needs of an organization), defining that goal in a tractable way into a set of quantitative questions that in turn define a specific set of metrics and data for collection. Furthermore, the tractability of this process allows the interpretation of the collected data and computed metrics in the appropriate context of questions and the original goal.

  The process of defining goals and refining them into quantifiable questions is complex and requires experience. In order to support this process, a set of **templates** for defining goals as well as deriving questions was developed. These templates reflect our experience from having applied the goal/question/metric paradigm in a variety of environments (NASA, IBM, AT&T, Burroughs). Different templates exist for defining (1) measurement goals, (2) process related questions and (3) product related questions. Goals are defined in terms of purpose, perspective and environment. Process related questions are formulated for identifying the quality of use, the domain of use, the cost of use, the effect of use and the feedback from use of a particular process. Product related questions are formulated for defining a product in terms of physical attributes, cost, changes and context, and evaluating it.

  (1) Goal Definition Template (principles M1, M2, M3, and M4):

  - **Purpose:**
    To (characterize, evaluate, predict, motivate) the (process, product, model, metric) in order to (understand, assess, manage, engineer, learn, improve) it. E.g. To evaluate the system testing methodology in order to improve it.

- **Perspective:**
  Examine the (cost, effectiveness, correctness, errors, changes, product metrics, reliability, etc.) from the point of view of the (developer, manager, customer, corporate perspective, etc) E.g. 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. E.g. The product is an operating system that must fit on a PC, etc.

(2) **Question Definition Templates:**

- **Process Questions:**
  For each process under study, there are several subgoals that need to be addressed. These include the quality of use (a quantitative characterization of the process and an assessment of how well it is performed), the domain of use (a quantitative characterization of the object to which the process is applied and an evaluation of the knowledge of the performers of the process concerning this object), cost of use (a quantitative characterization of the cost of performing each of the subactivities of the process) effect of use (a quantitative characterization of the output produced by the process and an evaluation of the quality of this output with respect to some quality or productivity aspect), and feedback from use (a quantitative characterization of the major problems with the application of the process so that it can be improved).

  Other subgoals involve the interaction of this process with the other processes and the schedule (from the viewpoint of validation of the process model).
- **Product Questions**
  For each product under study there are several subgoals that need to be addressed. These include the definition of the product (a quantitative characterization of the product in terms of physical attributes such as size or complexity, cost, changes and defects, and context such as customer community or operational profile) and the evaluation and improvement of the product with respect to a particular quality such as reliability or user satisfaction. Because the evaluation and improvement of a product is relative to particular quality aspects, its physical characteristics need to be analyzed relative to these quality aspects.

These templates 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 goal/question/metric models generated from these templates will be different from project to project and organization to organization. This reflects their being tailored for the different needs in different projects and organizations. It also acknowledges the need for different interpretation contexts in different environments (principle M8).

- **Construction** of the required products follows the guidelines defined as part of the planning activity in order to achieve quality a priori (principle P1); the existence of construction guidelines helps in assuring that methods are being used as intended (principle P3). It should be noted that the construction activity includes constructing the traditional project documents (e.g. requirements, design, code) as well as all other kinds of analytic information prescribed by the planning process (e.g., test results, scheduling data, effort data). The construction of analytic information is supported by data collection, data validation, and the computation of metrics as prescribed during the planning phase (principle M11).

- **Learning and feedback** is based upon a paradigm for evaluation and improvement (principle P4). The learning requires monitoring (measuring) the engineering and management processes as well as products (data), comparing the actual results (data) with the desired results, interpreting the results according to the context (principle M9) defined as part of the planning activity, and feeding the lessons learned back into the ongoing project (which might result in iterating the project plans) or into the planning phase of future projects. Feedback (principle P5) is important to engineers and managers. An effective feedback mechanism is especially crucial for supporting the complex management decision process (principle P10). The effectiveness of the feedback mechanism depends heavily on whether the appropriate interpretation context was provided for during the planning phase (principle M9) as well as amount, quality and accessibility of a body of experience (principles M12 and M14).

The presented abstract engineering process model built upon the principles in section 2 and 3 can be viewed as an improvement paradigm for software engineering allowing for the development of quality software as well as the evolutionary learning and accumulation of experience [3]. The need for integrating the building aspect and the measurement aspect is reflected in our software engineering process model.

At the University of Maryland we have been working to incorporate these principles into our work. We have developed an evaluation and improvement paradigm for the software development process and product, continued to formalize the goal/question metric paradigm to aid in developing an operational set of project and corporate goals for software development and provide a mechanism for evaluation and feedback, created an historical database for at least one environment (the Software Engineering Laboratory), experimented with expert system technology to help in the formalizing of expert knowledge, begun formalizing a mechanism for tailoring the process under controlled conditions, developed classification schemes for experimental analysis, developed meta-models that can be tailored for specific environments, and recently to provide automated support for all these measurement and evaluation activities via the measurement and evaluation system TAME (Tailoring A Measurement Environment). The TAME system development is part of the TAME project and is aiming at the integration of all the measurement principles presented in the previous section into SEEs.

5. Integrated Software Engineering Environments

The goal of an Integrated SEE (ISEE[*]) should be to support the planning, construction, and learning and feedback activities of a tailorable and tractable software engineering process. This includes support for goal-oriented measurement as a means for achieving tailorability and tractability.

SEEs cover a wide range of capabilities. We will characterize three classes of SEEs. At the minimal level an SEE can be a **set of tools** to support product development. A more sophisticated SEE would consist of an **integrated set of tools** that support one or more specific process models. The most sophisticated SEE would support all the activities required to **tailor any process model, method, and tool to a specific set of project needs**. The varying levels of sophistication require different degrees of support built into the SEE.

A high-level model of an ISEE is presented in figure 1. The original version of this model was developed during a panel session of the Workshop on "Requirements for Software Engineering Environments", held at the University of Maryland in May 1986 [58]. The objective of an ISEE is to support a software project producing engineering output objects (e.g. requirements document, design document, source code) and consuming engineering input objects (e.g. an informal requirements statement, quality requirements). The software project is conducted according to a particular software process model. The following components of an ISEE have been identified as crucial: (1) people, (2) methods and tools, and (3) a data repository (e.g. product library, measurement library).

People need to plan for a software engineering project by choosing a process model, a method or a set of methods for a particular set of input objects and a particular project goal set, based upon information about which ones work best in this particular environment. This information typically comes from the experience of the individual managers and engineers or some set of software engineering standards or process model set up by the organization. However in a more sophisticated environment it can come from a knowledge base of information, based upon the organization's experience in developing software. This implies some form of **data repository** of information ranging from a data base on the effectiveness of individual techniques, to a knowledge base dealing with the interconnectivity, tailorability and performance of different methods and tools in various organizational environments based upon specific project goals. Some of this experience can be incorporated in the **methods and tools** themselves. The construction of the output objects is supported by the prescribed set of **construction oriented methods and tools** Pieces from the **product library** might get reused for construction and output objects might be entered into the **product library** for reuse in future projects. The construction process as well as the produced output objects need to be monitored by **analytical methods and tools** All the measurement data taken and metrics computed are stored in the **measurement database** in order to increase the amount of information reflecting the organization's experience Data are interpreted based upon the information characterizing the particular project environment and interpretations of similar situations in similar projects. The amount of information representing historical knowledge is crucial for an effective interpretation and feedback mechanism. Feedback can result in learning by the project personnel and in changing the process model, **methods and tools**.

[*] We will use the term ISEE whenever appropriate; otherwise we use the term SEE.

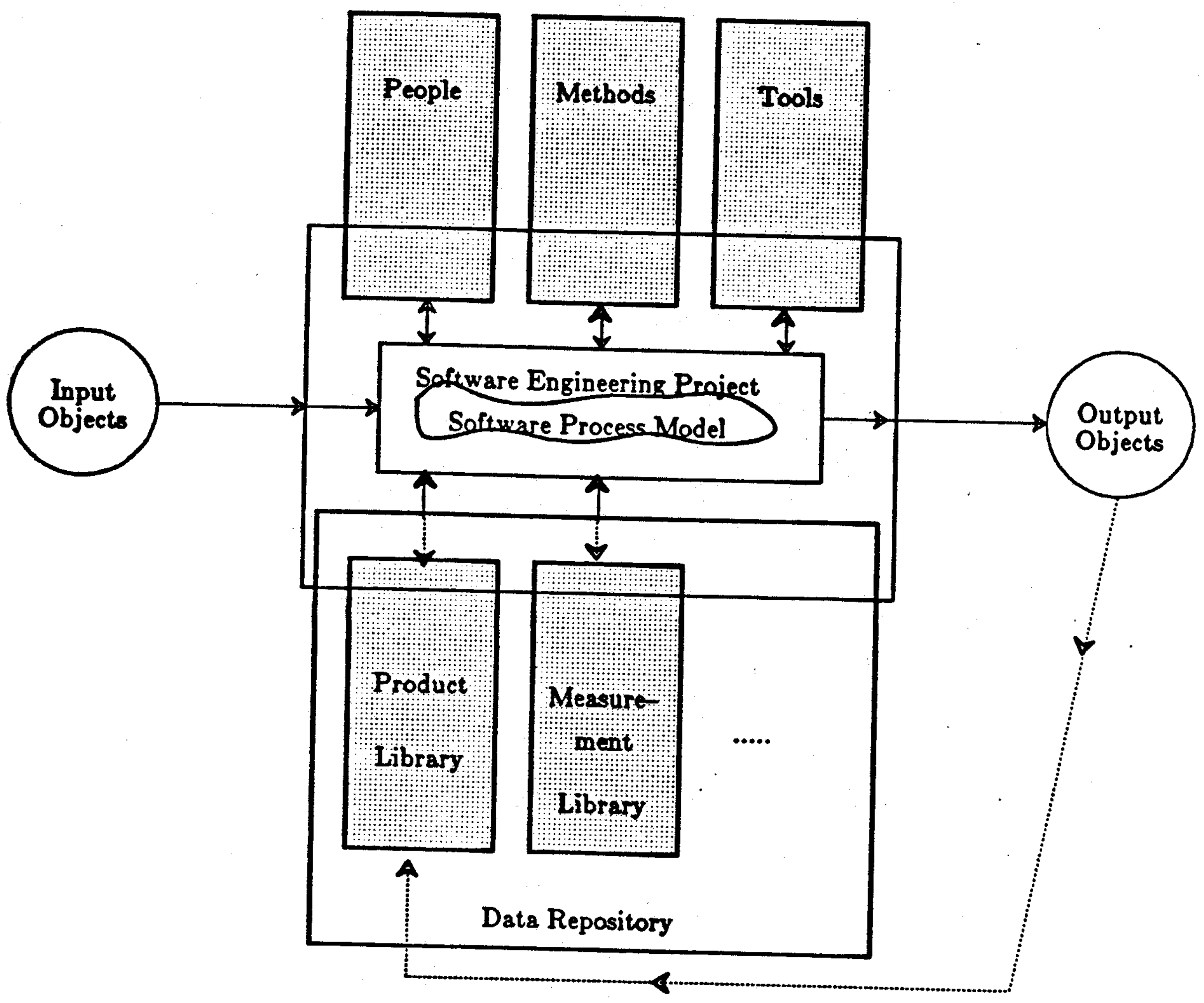


Figure 1: Model of an ISEE

All the information produced and consumed in an ISEE can be categorized according to three different schemes (see figure 2). According to the **organizational scheme** we might be interested in the effect on one particular project or across multiple projects within the organization. According to the **integrational scheme** we can support assessment, evaluation and improvement of individual methods or tools in isolation (local) or their integrated use in an overall process model. According to the **analytical scheme** we can provide support for assessment according to three categories, which we will refer to as measurement, feedback and planning. **Measurement** provides primitive data allowing for the quantitative characterization of process and product aspects. **Feedback** implies a higher level form of information based upon the interpretation of the measurement data in a context. **Planning** is based upon laying out the project goals, defining them operationally and interpreting them in context. For each of the categorization schemes the individual categories are listed in the order of increasing complexity: The assessment of process and product aspects from the point of view of an organization is based on their assessment in individual projects; the analysis of process and product aspects integrated into the overall process model requires their prior local analysis; and planning requires knowledge generated based upon feedback from prior projects via measurement.

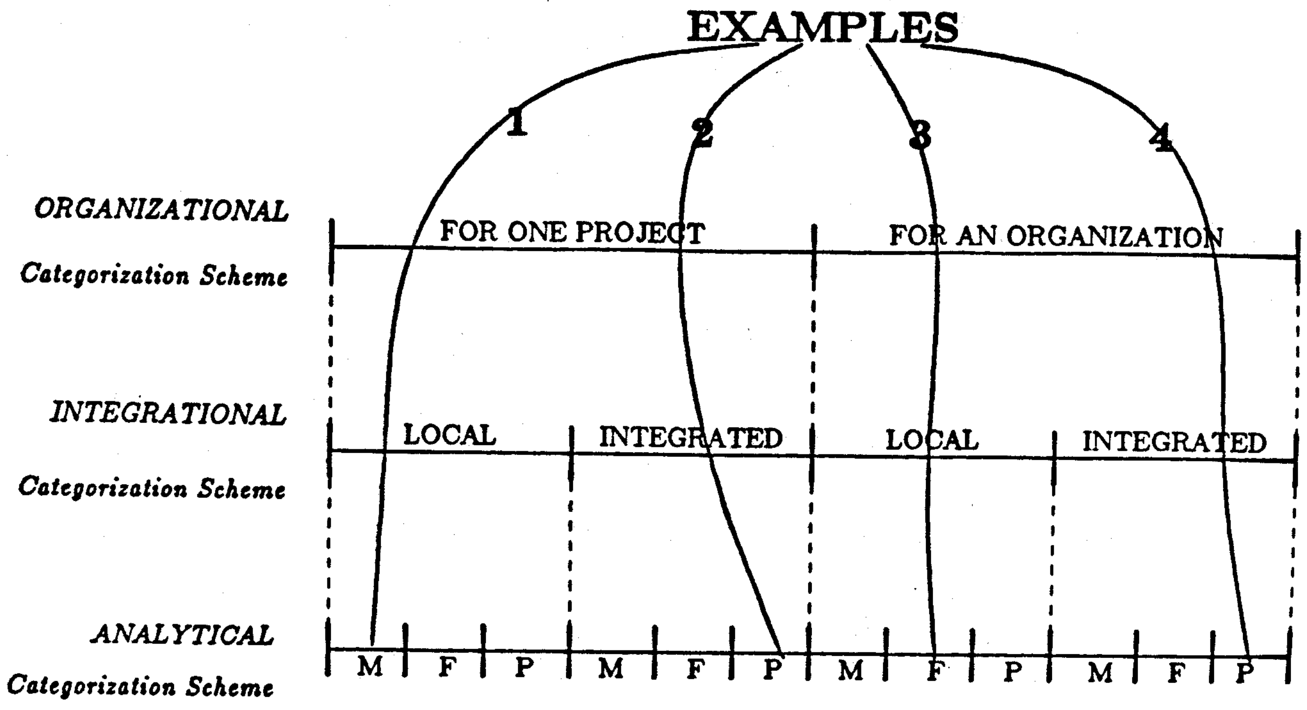
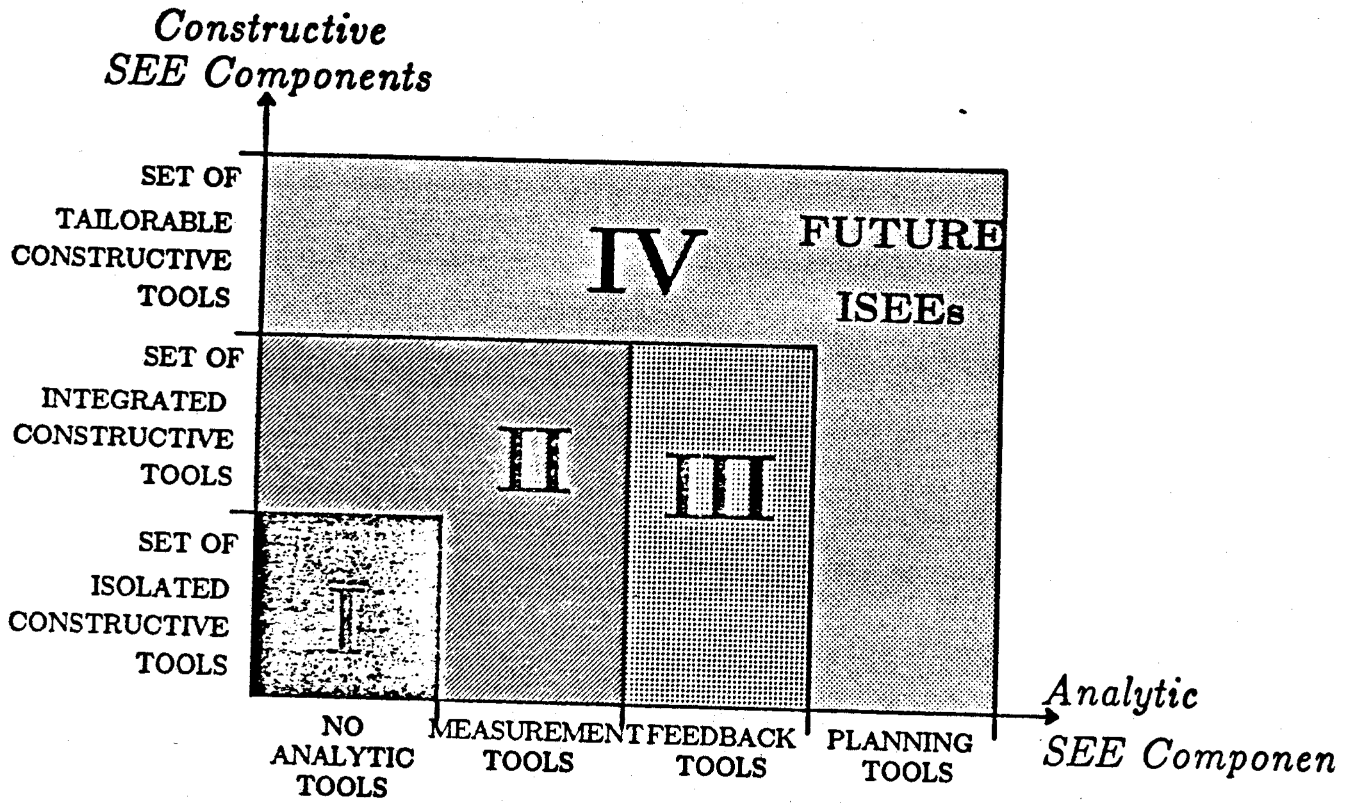


Figure 2: Categorization Schemes for Software Engineering Information

Four examples of possible combinations of the three information dimensions are given. At the simplest level, we may want to characterize the effect of a particular method (*local*) in the context of a given *project* using *measurement*. Specifically we may want to measure the number of failures discovered during system test or the complexity of the source code. At another level we may be interested in *planning* for the use of an *integrated* set of methods and tools to support the goals of a given *project*. Here we are interested in laying out project goals, refining the methods and tools based upon those project goals, measuring the effectiveness of the methods and tools toward achieving those project goals, and providing high level assessment back to the project manager on how well the particular instantiation of those methods and tools are working and how they should be modified during project development. Another example might be to understand the effectiveness of an individual method (*local*) across an *organization* based upon *feedback* from multiple projects within the organization. This might be to assess the effect of a system test tool used in different projects to evaluate it overall and possibly learn how to modify and refine it for specific applications. At the most sophisticated level, we want to *plan* for the tailoring of a process model and an *integrated* set of methods and tools for a class of projects within the *organization*. This might be used to determine if there is a specific configuration of a set of methods, tools and people for a class of projects, e.g. compiler development, common to the organization. This would permit the organization to learn how to provide the appropriate ISEE for any compiler development. The learning process takes place by specifying the characteristics of the project class, defining the relevant set of goals, specifying and applying the candidate methods and tools, measuring the effectiveness of the methods and tools based on those goals and interpreting within the context of the project goals, and providing feedback for the refinement and improvement of the environment.

Based on the different levels of support for construction oriented activities (isolated, integrated, tailorable) as well as the different ways for providing the needed information (measurement, feedback, planning), we classify SEEs by the degree to which they provide constructive and analytical support (see figure 3).



Constructive SEE Components
SET OF TAILORABLE CONSTRUCTIVE TOOLS
SET OF INTEGRATED CONSTRUCTIVE TOOLS
SET OF ISOLATED CONSTRUCTIVE TOOLS
IV
FUTURE ISEEs
II
III
I
NO ANALYTIC TOOLS
MEASUREMENT TOOLS
FEEDBACK TOOLS
PLANNING TOOLS
Analytic SEE Componen

Figure 3: Classification of SEEs

There is the environment type I consisting of a set of tools supporting construction. These SEEs do not allow for controlling the construction process nor do they allow for learning and feedback. The environment type II supports the construction by a set of integrated tools according to one process model and allows for the quantitative characterization of various process and product aspects via measurement tools. The environment type III supports feedback in addition to the capabilities of II. Feedback allows the understanding of the problems with process and product and their improvement as well as project specific learning. Only the environment type IV allows for tailoring the process model and the set of methods and tools to the specific needs of a project and the organization. This tailoring requires the planning of the specific project goals and environment characteristics and their operational definition as well as knowing the effect of candidate models, methods and tools in this situation. Learning of an organization is only possible if we can extract information from multiple projects. The needs of each project are unique; therefore, planning and recognizing the commonalities as well as the differences between projects is crucial to make an organization learn.

It is obvious that most commercially available SEEs are of class I. Some research projects include measurement and feedback mechanisms [35, 43, 52]; based on our information it is hard to decide whether they fall into class II or III. We know of no research projects addressing the planning issue.

With the TAME system development we attempt to contribute to the needed changes towards ISEEs of type IV. We provide automated support for measurement, feedback and planning.

6. The TAME System: A Software Measurement Environment

The TAME (Tailoring A Measurement Environment) system automates as much of the measurement process as possible, by supporting goal development into measurement via models and templates, providing evaluation and analysis of the development and maintenance processes, and creating and using historical data and knowledge bases that incorporate experience from prior projects. TAME will automate these measurement aspects within the framework of the analytical dimension of the ISEE model presented in the previous section. In this section we present the requirements for TAME, its architecture, and the scope of the first prototype. We have planned a series of prototypes being built using the iterative enhancement model. This approach is necessary because more research is needed in many areas (e.g., measurement, artificial intelligence, databases and systems) before the idealized TAME system can be built which will fulfill the entire set of requirements. As research results become available we will enhance our prototypes.

6.1. Requirements

The requirements for the TAME system can be derived from sections 4 and 5 in a natural way. These requirements can be divided into direct requirements (defined by and of obvious interest to the TAME user) and indirect requirements (defined by the TAME design team and required to support the direct requirements properly):

The first seven (direct) requirements include support for the planning activity by automating the goal/question/metric paradigm, for the construction activity by automating data collection, data validation and evaluation, and the learning and feedback process by automating interpretation and organizational learning. In addition, the user requirements concerning the TAME interface and the ability to produce appropriate reports are addressed.

(R1) A mechanism for defining measurement and evaluation goals in an operational and quantifiable way

We use the goal/question/metric paradigm and its templates for defining goals operationally and refining them into quantifiable questions and metrics. The selection of the appropriate goal/question/metric model and its tailoring need to be supported. The user will either select an already existing model without any changes 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 goal/question/metric paradigm and its templates can be formalized. The user needs to be supported in defining his/her specific goal according to the goal definition template. Based on this goal definition, the TAME system will search for a model in the data repository. 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 code reading method in the same environment, and from a model for assessing the same system test method in a different environment. A complete goal/question/metric model includes rules for interpretation of metrics and guidelines for collecting the prescribed data. As much of this information as possible will be generated by the TAME system automatically.

(R2) The automatic and manual collection of data and the validation of manually collected data

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 occurred (e.g. whenever a new product version is entered into the product library 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 not violated 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. It can be expected that the TAME system will be applied to software projects using different implementation languages. Using TAME across different language environments would require the replicated implementation of all these language dependent product measurement tools. A language independent product language needs to be defined in order to avoid this form of replication. Such a concept would allow us to develop one translator for each language allowing for the translation of products written in this language into the language independent representation, and one measurement tool for each metric. For n metrics and m languages, we would need to implement nxm measurement tools without the concept of an intermediate language, but only n+m tools using this concept.

(R3) A mechanism for controlling measurement and evaluation

In our case a goal/question/metric model specified the execution of a particular measurement and evaluation session control-wise. Executing a goal/question/metric model includes triggering 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 user needs to be informed.

(R4) A mechanism for interpreting analysis results in a context and providing feedback for the improvement of the process model, methods and tools

We use a goal/question/metric model to define the rules and context for interpretation of data and feedback for the purpose of refining and improving process 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 goal/question/metric 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 data are 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

The learning process is supported by applying measurement and evaluation to project classes of interest. For each of those classes, a historical database needs to be established concerning the effectiveness of the candidate process models, methods and tools. Feedback from ongoing projects of the same class, the corresponding process models, methods and tools can be refined and improved.

(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 mechanisms will be incorporated whenever useful and possible. The logical user interface is the user's view of measurement and evaluation. 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 goal/question/metric 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 to which this logical user interface can be used (= ease to which goal/question/metric models can be generated). Homogeneity is important for both the physical and logical user interface.

(R7) An effective mechanism for producing a variety of reports

The documentation of measurement, evaluation, and interpretation results in form of hard copies 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 goal/question/metric models based on which reports can be generated automatically. A laser printer and multi–color plotter would allow the appropriate documentation of tables, histograms and other kinds of textual and graphical representations.

The remaining five (indirect) requirements deal with the data repository issue, organizational issues such as mechanisms for security, access control and configuration management control, and system requirements for interfacing TAME with construction oriented SEEs and executing TAME on a distributed architecture. All these issues will are important in order to support all the direct requirements. Indirect TAME requirements are:

(R8) The effective storage and retrieval of all relevant information in a data repository

All data and knowledge required to support tailorability and tractability needs to be stored in a data repository. Such a data repository needs to be able to store goal/question/metric models, engineering products (product library in the SEE model) and all kinds of measurement data (measurement library in the SEE model). It needs to store data derived from the current project as well as historical data from prior projects. The effectiveness of such a data repository will be improved for the purpose of learning and feedback if, in addition to measurement data, interpretations from various evaluation sessions are stored and when interpretation rules will become integral part of such a repository. The data repository 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 data repository as an abstract data type allows us to use currently available database technology and substitute it later as more appropriate technology 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 [40].

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

TAME needs to control the access of users to the TAME system itself, to various system functions and the database. These are typical functions of a security control system. The actually enforced security strategies depend on the project organization. It is part of planning a project to decide who needs to have access to what function and what piece of information. 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 goal/question/metric 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 the related measurement data and the goal/question/metric model according to which those measurements were taken. TAME needs to make sure that a user always knows whether data in the repository are consistent with the current product version and were collected and interpretated according to a particular model. The actual configuration management and control strategies will result from the project planning phase.

(R11) An interface to construction oriented SEEs

On one hand, it might be necessary to collect data (e.g. the number of activations of a compiler, the number of test runs) from the actual development or maintenance process or have access to the products. On the other hand, our mid-term goal asks for interfacing TAME with SEEs for the purpose of on-line feedback into ongoing development or maintenance activities. Models for appropriate interaction between constructive and analytic processes need to be specified. Interfacing with construction oriented SEEs poses the problem of interconnecting systems implemented in different languages and running on different machines (with probably different operating systems). in an efficient way.

(R12) A structure suitable for distribution

TAME will ultimately run on a distributed system consisting of at least one main-frame computer and a number of workstations. The main-frames are required to host the data repositories which can be assumed to be very large. The rest of TAME might be replicated on a number of workstations.

6.2. Architectural Design

The TAME architecture in figure 4 describes the individual components of the TAME system and their interrelationships. According to the SEE model presented in section 5, the architectural components are tools and a data repository.

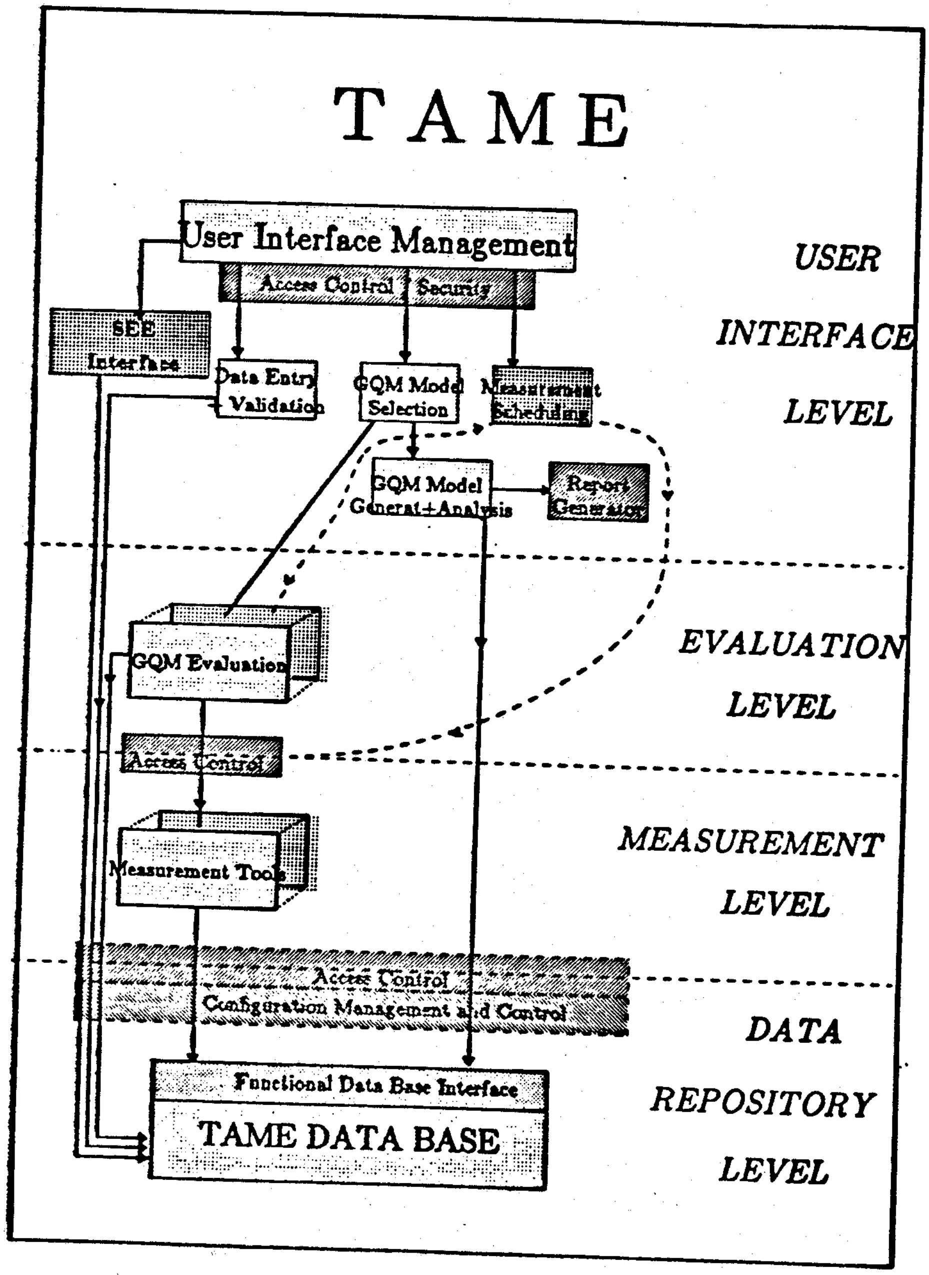


Figure 4: Architectural Design of the TAME System

We group the TAME components into four logical levels, the user interface, evaluation, measurement and data repository level. Each of these four TAME levels consists of one or more architectural components:

• **The User Interface Level** consists of the User Interface Management Tool, one of the Access Control Tools, the Data Entry and Validation Tool, the GQM Model Selection Tool, the GQM Generation and Analysis Tool, the Report Generating Tool, the Measurement Scheduling Tool, and the SEE Interface Tool.

• **The Evaluation Level** consists of the GQM Evaluation Tool.

• **The Measurement Level** consists of a set of Measurement Tools and one of the Access Control Tools.

• **The Data Repository Level** consists of one of the Access Control Tools, the Configuration Management and Control Tool, and the Data Repository.

In the following we discuss the relationship between the TAME requirements and the architectural components as well as the interrelationship between these components.

The User Interface Level The User Interface Tool implements the physical user interface. The SEE Interface Tool takes care of the interaction between TAME and construction oriented SEEs. The inputing of non-automatically collected data and their validation is implemented by the Data Entry and Validation Tool. The logical user interface is implemented by the GQM Model Selection Tool; this tool guarantees that no access to the evaluation, measurement or data repository level is possible without using a goal/question/metric model. In addition, the user interface level contains the GQM Model Generation and Analysis Tool for generating goal/question/metric models and the Report Generator Tool for producing all kinds of reports. Finally, the Measurement Scheduling Tool triggers data collection via measurement tools according to planned schedules.

The Evaluation Level This subsystem performs the evaluation according to a particular goal/question/metric model. In addition, the GQM Evaluation Tool needs to know the specific authorizations of the user in order to know which evaluation functions can be performed by this particular user. The GQM Evaluation Tool also provides analysis functions, for example, telling the user whether certain measures can be computed based upon the data currently available in the data repository. This analysis feature of the subsystem is used during the creation phase of goals, questions, and measures, as well as during the actual evaluation phase according to previously established goals, questions, and metrics.

The Measurement Level The Measurement Level consists of tools for computing metrics.

The Data Repository Level The Data Repository Level provides the infrastructure for various types of evaluation. This level allows storing and retrieving all kinds of software related data. In addition, the Configuration Management and Control Tool is viewed as part of or interface to the data repository level. Data can only be entered into or retrieved from the data repository under configuration and management control.

Orthogonal Access Control Component The Access Control Component is orthogonal to the four level structure of TAME. It consists of a number of tools distributed across the logical architectural levels and are therefore discussed separately.
The TAME Access Control Component consists of three tools. One tool validates access to the TAME system itself and to various functions at the user interface level based upon the rights assigned to a particular user. The two other data access control tools control access to various

measurement tools and access to the database.

6.3. First TAME Prototype

The first of a series of prototypes has been developed for supporting measurement in Ada projects [18]. This first prototype implements all four logical levels of the architecture. However, the automated support for some of the activities falls short of the requirements stated in section 6.1 because the state-of-the-art did not provide for their implementation. The choice of Ada does not effect the TAME prototype except for the measurement tools which need to be run on Ada source code.

The first prototype enables the user to generate goal/question/metric models using a structured editor. Existing models can be selected by using a unique model name. No support for selecting models based on goal definitions or for reusing existing models for the purpose of generating new models is offered. Evaluation sessions can be run according to existing goal/question/metric models. However, no support for interpretation is provided. Metric values 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 consists of three tools for computing product measures from Ada source code: a static source code analyzer computing all basic source code counts including lines of code, frequency of use of particular language features, cyclomatic complexity metrics and software science metrics, a data bindings analyzer, and a structural coverage analyzer [56]. Similar tools for conventional languages such as Fortran are exist too [33]. A general schema for a software engineering data repository has been developed and implemented [40]. The current implementation is based upon the relational database system from ORACLE Corporation.

The first prototype is running on a SUN-3 under UNIX. It is implemented in Ada (as far as the language dependent measurement tools are concerned) and C.

More research is needed before the idealized TAME system can be built. Major areas of research include measurement, databases, artificial intelligence, and systems. Specific high-priority topics are a formal language for specifying goal/question/metric models, the definition of more and better models, mechanisms for better tailoring and reusing project knowledge, mechanisms for better interpreting metrics in the context of questions and goals, component interconnection languages, a language independent representation of software, better mechanisms for data access control and configuration management control, software engineering database definitions, and distributed system architecture. As results become available we will integrate it into an enhanced prototype.

7. Summary and Conclusions

Based upon a dozen years of analyzing software engineering processes and products, we have proposed a set of software engineering process and measurement principles. These principles have led us to recognize the need for the software engineering process to support multiple process models across the full software life cycle. Such an environment must support not only the engineer but the manager. It must combine the technical and managerial aspects of software engineering. The full software process life cycle consists of three stages: planning, construction, and feedback and learning. The planning phase consists of the establishing of goals specific to the project as well as the organization, and the selection of the process model, methods and tools appropriate to those goals. The construction process consists of the the development or maintenance of the product and the analysis of the process and product relative to the goals set. The feedback and learning process consists of project tracking via a mechanism that provides information for improving the current project as well as future projects.

Based upon this definition, the software engineering process need to be tailorable and tractable. We need the ability to tailor the process, methods and tools to specific project needs in a way that permits maximum reuse of prior knowledge. 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 address all of these issues.

We have argued that the tailorability and tractability attributes of the software planning, construction and feedback and learning processes require the support of a measurement process. The measurement process needs to be top–down, based upon operationally defined goals.

The TAME project uses the goal/question/metric paradigm to support this type of measurement paradigm. It provides for the establishment of project specific goals and corporate goals for planning, provides for the tracing of these goals throughout the software life cycle via feedback and post mortem analysis, and offers a mechanism for long range improvement of all aspects of software development.

The TAME system automates as much of this process as possible, by supporting goal development into measurement via models and templates, providing evaluation and analysis of the development and maintenance processes, and creating and using databases of historical data and knowledge bases that incorporate experience from prior projects.

The short range (1-3 years) goal for the TAME system is to build the evaluation 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 system 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 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, data bases, artificial intelligence, and systems. Specific activities needed to support TAME include more formalization of the goal/question/metric paradigm, the definition of better models for such attributes of quality and productivity, mechanism for better formalizing the reuse and tailoring of project knowledge, the interpretation of measures 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.

It has been our plan to form a collaborative arrangement with several companies, government agencies and other research institutions in order to seek support in the development of the TAME system. The support has come in two ways: financial and effort. To be a collaborator in the TAME project, companies support the development of some specific tool or subsystem. The financial support is used to research and build the particular tool or subsystem. It entitles the organization to input into the prototyping of that specific tool or subsystem as well as have access to all aspects of the TAME system. The effort support by the organization is in the productizing of the tool or subsystem. The product belongs to the organization, but is available to the University of Maryland for further work on the TAME system.

8. Acknowledgements

We want to acknowledge the many contributions to the TAME project and, thereby indirectly to this paper, by Michael Daskalantonakis, Alexis Delis, Dennis Doubleday, Leo Mark, Karl Reed, P. Dave Stotts, A. Joe Turner, Shouli Wang, Linda Wu, and Shi Xiao-Hong.

9. 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," Proc. of the Fifth International Conference on Software Engineering, San Diego, USA, March 1981, pp. 107–116.

[3] 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].

[4] 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, December 1985.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] V. R. Basili, D. H. Hutchens, "System Structure Analysis: Clustering with Data Bindings," IEEE Transactions on Software Engineering, August 1985, pp. 749–757.

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

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

[13] V. R. Basili, C. Loggia–Ramsey, "ARROWSMITH–P: A Prototype Expert System for Software Engineering Management, Proc. of the IEEE Symposium on Expert Systems in Government, October 23–25, 1985, pp. 252–264.

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

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

[16] V. R. Basili, J. Ramsey, "Structural Coverage of Functional Testing," Proceedings of the Eighth International Conference on Software Engineering, London, UK, August 1985.

[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, "TAME: Tailoring an Ada Measurement Environment," Proceedings of the Joint Ada Conference, Arlington, VA, March 16-19, 1987, pp. 318-325.

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

[20] 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.

[21] V. R. Basili, R. W. Selby, Jr., "Comparing the Effectiveness of Software Testing Strategies," Technical Report TR-1501, Dept. of Computer Science, University of Maryland, College Park, May 1985.

[22] 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.

[23] 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.

[24] 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.

[25] V. R. Basili, D. M. Weiss, "A Methodology for Collecting Valid Software Engineering Data," IEEE Transactions on Software Engineering, vol. SE-10, no.3, November 1984, pp. 728-738.

[26] P. A. Bernstein, "Database System Support for Software Engineering," Proceedings of the Ninth International Conference on Software Engineering, Monterey, CA, March 30 - April 2, 1987, pp. 166-178.

[27] D. Bjorner, "On the Use of Formal methods in Software Development," Proceedings of the Ninth International Conference on Software Engineering, Monterey, California, March 30 - April 2, 1987, pp. 17-29.

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

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

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

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] M. Dowson, "ISTAR – An Integrated Project Support Environment," Proceedings of the Second ACM Software Engineering Symposium on Practical Development Support Environments, ACM Sigplan Notices, vol. 2, no. 1, January 1987.

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

[37] 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.

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

[39] 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.

[40] L. Mark, H. D. Rombach, "A Meta Information Base for Software Engineering," submitted to IEEE Transactions on Software Engineering.

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

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

[43] L. Osterweil, "Software Processes are Software Too," Proceedings of the Ninth International Conference on Software Engineering, Monterey, CA, March 30 – April 2, 1987, pp. 2–13.

[44] 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.

[45] 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.

[46] C. Loggia-Ramsey, V. R. Basili, "An Evaluation of Expert Systems for Software Engineering Management," Technical Report TR–1708, Department of Computer Science, University of Maryland, College Park, MD, September 1986.

[47] 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.

[48] H. D. Rombach, "A Controlled Experiment on the Impact of Software Structure on Maintainability," IEEE Transactions on Software Engineering, vol. SE–13, no. 3, March 1987, pp. 344–354.

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

1987.

[50] 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.

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

[52] 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.

[53] R. W. Selby, Jr., V. R. Basili, and T. Baker, "CLEANROOM Software Development: An Empirical Evaluation," Technical Report TR–1415, Dept. of Computer Science, University of Maryland, College Park, February 1985 [is accepted for publication in IEEE Transactions on Software Engineering].

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

[55] Webster's New Collegiate Dictionary, G + C Merriam Company, 1981.

[56] 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.

[57] 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.

[58] M. V. Zelkowitz (ed.), Proceedings of the University of Maryland Workshop on 'Requirements for a Software Engineering Environment', Greenbelt, MD, May 1986, Technical Report TR–1733, Dept. of Computer Science, University of Maryland, College Park, December 1986.