Measurement-Based Guidance of Software Projects Using Explicit Project Plans

Christopher M. Lott and H. Dieter Rombach
Arbeitsgruppe Software Engineering
Fachbereich Informatik
Universität Kaiserslautern
67653 Kaiserslautern, Germany
Email: lott@informatik.uni-kl.de,
rombach@informatik.uni-kl.de

Appeared in *Information and Software Technology*,
Volume 35, Number 6/7, June/July 1993.

Abstract

As first steps towards establishing software engineering as an engineering discipline, we need to create explicit models of its building blocks, i.e., projects, processes, products, and various quality perspectives; organize these models for effective reuse across project boundaries; and establish measurable criteria for project guidance. This paper investigates the possibilities of providing measurement-based project guidance using explicit project plans. Following a summary of technologies developed by the process modeling and measurement subcommunities of software engineering, a method for integrating these technologies is suggested, and the potential benefits for project guidance are discussed. Examples from the MVP Project at the University of Kaiserslautern are used throughout for illustration purposes.

Keywords: improvement-oriented engineering model, measurement, explicit models, project modeling, project guidance.

1 Introduction

Software development and maintenance projects are difficult to plan and manage [1]. Major reasons include the lack of explicit process models, the lack of operational definitions of target qualities, and the lack of tractable quality models.[1] Major strides towards introducing more engineering discipline into the processes of software evolution (i.e., development and maintenance) have been made by the software engineering community. The process modeling and measurement subcommunities especially have accumulated a significant body of knowledge and provided promising technologies [2, 3]. The purpose of this paper is to demonstrate how integrating explicit process models and measurement can offer the needed intellectual control over software evolution projects.

Process modeling and measurement technologies offer many possible avenues towards solving the problems of planning and managing projects. Process modeling technology is needed to capture project goals explicitly; to build explicit models of existing, real-world processes; to integrate these models with other elements into comprehensive project plans; to guide and control project performance using explicit project plans; to enable changes in a process to be be specified, reviewed and planned; and to improve project plans both on-line (i.e., while a project is being performed) and off-line (i.e., for future projects). A number of pro-

[1] An *operational* quality definition is an explicit, objective definition that allows unambiguous differentiation between different instances of the quality aspect in question. An operational definition typically requires measurement. A *tractable* quality model is an operational definition which allows tracing of quality aspects through all relevant stages of a project.

cess modeling languages have been developed [4], and first experiences regarding their practical usefulness are documented [5, 6, 7, 8, 9].

Measurement technology is needed to define target data values for the qualities to be achieved by a project, to make the status of a project visible, to provide feedback that guides the performance of project team members, to support prediction of future project performance, and to establish baselines against which improvement claims can be judged. A number of measurement approaches have been developed based on the idea of tying measures to project goals; one example is explained in in [10, 11]. Experience tells that useful measurement plans can be established and significant improvements can be achieved based on goal-oriented measurement approaches in local environments [12].

There is a growing understanding that useful project plans must be based on explicit models of processes, products, quality aspects, etc. The best process models describing "how" a process should be performed are useless without an operational, measurable description of "how well" it should be performed, defined in terms of expected characteristics of the resulting products and the processes themselves. The reverse is also true; the best measurement plans are useless if they are based on incorrect assumptions regarding the performance of the projects to be measured [9]. We suggest explicitly integrating process modeling and measurement technologies.

This paper describes how software evolution teams can gain intellectual control over their projects by modeling a project explicitly, instrumenting it with measurable target values according to empirical quality models, guiding the project according to the instrumented and explicit project plan, collecting measurement data, and quantifying process improvements. We describe an approach for improvement-oriented software engineering that is suited for integrating process modeling and measurement, survey existing process modeling and measurement technologies, propose a method for integrating process modeling and measurement technologies, and describe how tractable project plans can be used to guide projects. Examples from the MVP Project at the University of Kaiserslautern are used throughout the paper for illustration purposes.

2 Improvement-Oriented Software Engineering

Software engineering involves the development and application of explicit models of the basic components of the discipline (e.g., processes, products, resources, qualities) in order to improve the understanding, planning, and guidance of real-world software projects. One of the major lessons learned from many years of software engineering experience is that such models are complex and need to be tailored and adjusted for changing project goals and characteristics [13]. Therefore, systematic software engineering requires explicit support both for continuously building new models and for improving existing ones based on lessons learned from ongoing projects. It also requires support for guiding ongoing evolution projects effectively based on the reuse of existing models. The use of measurement is essential in order to support learning from ongoing projects as well as to support guidance of projects based on sound, objective criteria.

2.1 The TAME Model

The Quality Improvement Paradigm (QIP) developed in the TAME project combines support for model building with support for project guidance, and also integrates measurement [10, 14]. Process improvement is achieved by repeating the following sequence of steps for each project:

- Characterize: The project at hand is characterized based on available models for similar projects.
- Set goals: Project and quality improvement goals are identified and stated in a operational way.
- Choose models: A suitable project plan is devised in order to achieve the stated goals. This involves the selection of suitable models and metrics suited for measuring adherence to project goals. The quality of the plan depends on the maturity of the organizational set of capabilities.
- Perform project: The project is performed according to plan, data are collected, and project-specific feedback for planning and management is provided.
- Analyze: The project is assessed upon completion and lessons thereby learned are extracted for future

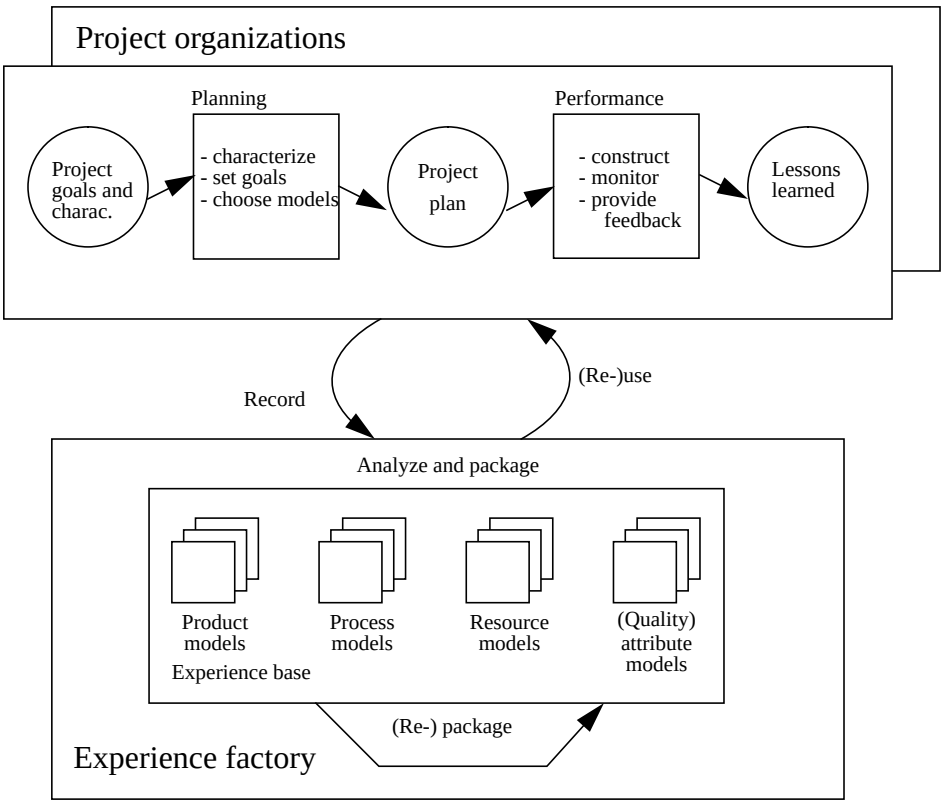

Figure 1: Improvement-oriented organizational structure

projects.

- Package: Based on feedback from each project, existing models are reconsidered. If needed, models are excluded, added, modified, or repackaged in order to improve their effectiveness for future projects.

Figure 1 depicts an improvement-oriented structure for a software organization based on the QIP. This structure divides responsibility for QIP steps between project-specific entities (i.e., project organizations), and the Experience Factory organization. The Experience Factory is solely in charge of preserving and repackaging models of existing experience. Both intra-project and inter-project feedback are supported by recording and reuse activities. All experience accumulated in a project, including measurement data, may be recorded. Reuse of existing models takes place during the planning stage of a project (where suitable models are chosen for given project goals and characteristics) as well as the performance stage of a project (where actual data are compared to historical baseline models in order to achieve guidance). Re-packaging refers to all activities aimed at improving the reuse potential of existing experience (includes generalizing, tailoring, formalizing) [14].

In the remainder of the paper we will use the following terminology:

- Descriptive (or "as-is") modeling: the activities aimed at explicitly capturing some aspect(s) of an existing software project.

- Prescriptive (or "to-be") modeling: the activities aimed at creating an explicit model of some aspect(s) to be achieved or followed in a future project.

- Performance tracking and guidance: the activities aimed at comparing actual project performance with prescriptive project plans in order to recognize deviations, to suggest alternate performance patterns for achieving project goals, and to initiate replanning activities.

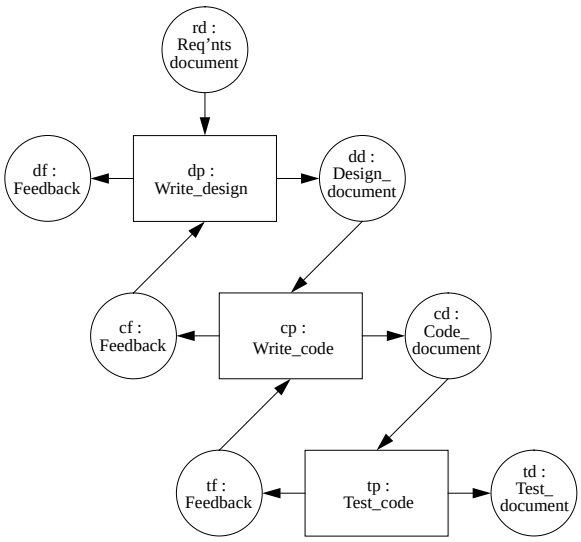

Figure 2: Product flow in DCT1 example

2.2 Example project plan

The following descriptive process model, abstracted and simplified from [15], is used throughout the paper. The product-flow view of this example appears in Figure 2. A development project, named the DCT1 project, is modeled as consisting solely of the three processes "Write_design," "Write_code," and "Test_code." Each process consumes and produces a single product document. Forward product flow is modeled as a single document produced by each process and consumed by its logical successor. Feedback among processes is produced and consumed similarly, but in the opposite direction. Control flow is modeled as iterative cycles through the set of processes. The environment conditions in which this project was performed are as follows: the technical experience of the development team was average for the environment (4 years); the implementation language was Ada; the application domain was satellite control systems; the reliability requirements on the software were average for the environment (0.5 faults per developed KSLOC was acceptable); and the primary goal was to deliver the product on schedule.

2.3 Descriptive Modeling

Descriptive modeling aims to improve the understanding of some aspect of a real-world software project. Starting from some partial or low-confidence models, we instrument a software project in order to improve these models. For example, if we have only limited understanding of how resources are used throughout the DCT1 project, we could instrument its project plan in order to capture resource data (e.g., amount of effort in staff hours per process of the project plan). Assume that post-project analysis of the collected measurement data shows that resources were used at a rate of 30% for writing design, 40% for writing code, and 30% for testing. Similarly, assume that post-project analysis of error data shows that code faults were detected at the rate of 0 in writing design, 4 per KSLOC in writing code, and 2 in testing. These characteristic numbers form simple quantitative models which can be used as baselines in future projects.

2.4 Prescriptive Modeling

Prescriptive modeling involves choosing the appropriate project plan from a library of such plans and choosing the appropriate supporting tools when given a set of project goals and characteristics (see also Figure 1). This requires that project goals and characteristics are defined in a measurable way, and that candidate models are available in the experience factory together with information on their effectiveness relative to the project goals and characteristics in question [14]. Consider an example instantiation of the descriptive project plan described above for a second project called DCT2, in which the characteristics match those of DCT1. The goals of DCT2 are to use not more than 1,000 staff-hours and to stay within the characteristic fault detection rate of project DCT1. Thus for the writing code activity, not more than 400 staff-hours (40% of 1,000) should be expended. This example instantiated a project plan without any changes. A more realistic case is that the goals of the project do not have exact matches in the library of models, which is the general reuse problem [14].

2.5 Project Tracking and Guidance

Project tracking is a prerequisite for a posteriori detection or a priori prevention of project performance problems, and for recognizing the need for replanning. Project guidance consists of suggesting alternate performance patterns to avoid reaching undesirable states. Assume that the DCT2 project has completed their writing code process but has exceeded their resource limit by 50 staff-hours and has detected only 3 faults per KSLOC instead of the expected 4. In this case, there are three

possibilities. First, the project can ignore the fact that they have violated the expectations as expressed in the project plan and continue. Second, the team can force a redo of the code process. Although the resource limit can never again be satisfied, perhaps additional unit testing will detect more faults. Third, the team can replan the project; essentially this recognizes that the plan as stated is unattainable. In this example, deviation from the project plan was detected only after a failure state was reached. It is desirable to predict possible plan violations as early as possible and take preventive actions before a failure state is reached. In the example given above, the deviation could have been predicted at the very moment when the resource limit was exceeded.

3 Existing Process Modeling and Measurement Technology

This section presents lessons learned about process modeling and measurement technologies and introduces the specific technologies developed and used in the MVP Project.

3.1 Existing Process Modeling Technology

Process modeling notations share a common purpose of building models of real-world processes, but otherwise differ widely. Some were designed to automate processes to the extent possible and others to aid understanding. Numerous examples of notations for building formal process models are surveyed in [2]. Lessons learned over the past five years of process research [2, 16] include the following:

- Descriptive versus prescriptive models: Processes already exist in software evolution organizations, so it is natural to begin modeling them. Starting from the current situation establishes the necessary baselines against which any future improvements must be measured. The case for descriptive modeling is consistent with the initial characterization step of the Quality Improvement Paradigm as discussed above. In the past, too little emphasis has been placed on descriptive modeling.

- Goal-oriented versus algorithmic models: To guide a project, descriptions of process and project goals (what should be achieved) and algorithms (how the goals can be achieved) are needed. Project goal definitions help in selecting the appropriate processes to match the given goals and also motivate project personnel in terms of what is expected. Algorithms are used to guide the performance of project members in terms of what they should do. With the exception of [17], too little emphasis has been placed on explicitly modeling project and process goals.

- In-the-large versus in-the-small models: Projects are typically performed by teams of developers. We distinguish between process modeling in-the-small (describing the process aspects concerning an individual team member) and process modeling in-the-large (the interfaces between different team members). Experience tells that the most leverage for improvement can be expected from a better understanding of the interfaces between team members. In the past, too much emphasis has been placed on modeling in-the-small.

- Off-Line versus on-line support: A project plan can be used off-line, as a reference document, or on-line, as the basis of a support environment to guide the performance of individual team members. Simply capturing and understanding process goals and measurement plans off-line can be helpful for teams, while on-line support can automate data collection and project tracking activities. We feel that too much emphasis has been given to on-line support of fully automatable tasks, at the expense of understanding deeper issues of team coordination and process guidance to provide suitable support for creative evolution activities.

- Guiding versus controlling projects: A variety of interaction models between team members and explicit project plans have been proposed for on-line use of such plans. The fundamental question is whether the human activity is controlled or guided. Creative tasks must remain under the control of the user, but guidance from a process-centered environment is useful. Mechanizable tasks can be controlled by the project plan; for example, by fully automating data collection from software artifacts. Guidance has received little emphasis in the past, in part because of the focus on the mechanizable, back-end activities of evolving programs such as compiling and linking.

- Static versus dynamic project plans: Projects are performed by humans, which means that project performance is inherently nondeterministic. Software projects are actually infinitely nondeterministic, meaning that it is impossible to model a finite set of plausible alternatives for the project that will be sufficient to perform the work successfully. There will always be unforeseen problems that must be handled during project performance. Therefore, static (unchangeable) project plans are not acceptable for guiding software projects. Little previous work (again an exception is [17]) has addressed the need for changeable plans.

3.1.1 The Process Modeling Language MVP-L

In the MVP Project, we developed the process notation called MVP–L [18]. Our notation was designed for modeling processes in-the-large, including building comprehensible specifications and designs of processes, products, and resources, and supporting the instantiation and execution of these models for the purposes of analysis, simulation, and project guidance. MVP–L project plans are composed of elementary process, product, resource, and attribute building blocks. Goals of a project are specified for individual processes using entry and exit criteria (rules). Algorithms of a project are optionally given for individual processes using an algorithmic language. Execution of MVP–L project plans for project simulation and performance guidance is based on the notion of project state and state transitions.

3.2 Existing Measurement Technology

Measurement is aimed at capturing some aspect of a software object in a quantitative form. In the past, the word "measurement" was synonymous with "metrics." Only recently have we learned that metrics need to be defined and interpreted in the context of a project environment. Lessons learned over the past 15 years of measurement research include the following:

- Measurement involves the specification, collection, and interpretation of measurement data for various purposes.
- Measurement is oriented towards better understanding (building baselines), better planning and management (prediction), and better quality and productivity (improvement).
- Various improvement approaches exist, include Deming's Plan-Do-Check-Act approach [19], Basili's Quality Improvement Paradigm [10], and the SEI's Capability Maturity Model (CMM) [20].
- There is no generally accepted set of process metrics, because processes, perspectives, roles, needs, and expectations vary too much.
- Owners of to-be-measured processes should be involved in defining goals of measurement, metrics, and interpretation of data (i.e., model building).
- We are capable of building models with strong predictive powers for local environments [12, 21].
- We are only starting to understand the impact of environmental factors on empirical models [12, 21].
- Measurement needs to be top-down or goal-oriented; objectives determine the choice of metrics [10].
- Various goal-oriented measurement approaches exist, including Murine and McCall's SQM [22], Akao's QFD [23], and Basili's G/Q/M [10, 11].
- Empirical research studies need to be based on sound principles [24].
- Process measurement requires an explicit model of the process object to be measured [6, 9].
- Process data are owned by individuals and only explicitly released for well-specified purposes.

A comprehensive state-of-the-art review of measurement and empirical research appears in [16].

3.2.1 The Goal/Question/Metric Paradigm

The Goal/Question/Metric paradigm (G/Q/M), which provides a framework for constructing a measurement plan and interpreting measurement results [10, 11, 25], is used in the MVP Project. G/Q/M supports both a top-down, definitional approach, in which goals are refined in a traceable way into metrics, and a bottom-up, interpretational approach, in which measurement data are interpreted in the context of the high-level goals. In the top-down approach, high-level goals are recursively refined into questions, and those questions are in turn refined into a series of atomic metrics. In the

bottom-up approach, measurement data gathered for the chosen metrics is used to answer the questions posed for each goal, and the answers are used to satisfy the goals. It is possible to validate the completeness of a set of metrics before data collection begins based on whether all questions can be answered. Additional features of this measurement paradigm is that people interested in achieving the goal can own and influence the measurement plan, and that the privacy of the data gathered to meet the goal can be preserved. An example of a high-level goal is "to analyze the cost-effectiveness of the unit-test process in order to understand it from a project manager's viewpoint in the organization's development unit."

4 Integrating Modeling and Measurement Technologies

Quality models are an integral part of any real-world project plan. In our opinion, a major problem with most process modeling notations is that they fail to reflect the necessary synergy between process modeling and measurement aspects. Most notations lack the facilities to describe how quality models relate to project goals and process algorithms, how measurement data can be collected, and how measurement-based analysis results can be fed back to guide project performance. We describe related work as well as the specific approach for integrating modeling and measurement technologies that was developed in the MVP Project.

4.1 Related Work

Research activities in the past have established measurement plans without process models and have built process models without incorporating target measurement values. However, recent trends emphasize the integration of these technologies; examples include the SEI's Capability Maturity Model (CMM) [26] and the most recent ESPRIT Call for Proposals [27]. The CMM combines process modeling with measurement of projects, products, and processes. The ESPRIT CFP calls for integration of processes and measurement to build process-centered environments. Currently, there seems to exist general agreement on the following three issues:

- Systematic planning and performance tracking of projects is only possible when project goals are defined objectively.
- Processes are specialized for each project's needs [10]. Needs in this sense may be any requirement imposed on the project organization, including product properties such as reliability and process properties such as resource consumption.
- Metric collection efforts depend on a sound, objective, unambiguous understanding of the processes from which data is collected [6, 9].

There is no general agreement as to how metrics and quantitative models of quality aspects should be technically integrated into process models.

4.2 In the MVP Project

We have developed a measurement-based process modeling technology by integrating the G/Q/M measurement technology and the MVP–L process modeling technology at the level of measurable data items. In MVP–L, measurable data items are represented as process, product, and resource object attributes, and in G/Q/M as metrics at the base of a G/Q/M plan. An MVP–L project plan can be integrated with a G/Q/M measurement and quality assessment plan by mapping G/Q/M metrics onto MVP–L attributes. These MVP–L attributes are used to connect with measurement tools for data collection during project performance and to define entry/ exit criteria (i.e., goals) of the project and individual processes. The eventual collection of data as demanded by the project plan and the evaluation according to quality criteria set out in the measurement plan can be accomplished in the context of a process guidance system.

A simple example of using our technology to integrate modeling and measurement activities is given next, using the DCT1 project and a G/Q/M measurement plan that measures the quality aspect of detecting defects in the requirements document. The example G/Q/M-goal is "to analyze the detection of requirements defects in order to understand it from the development team's viewpoint in the development organization." This goal is refined into questions that map directly into metrics, namely the number of requirements defects that were detected in the three processes of writing requirements, writing code, and testing code. To integrate these views, each metric is mapped onto an MVP–L attribute of the process models. Both how and when metric data should be collected and how the data should be interpreted in

the context of the process are thereby specified explicitly.

Next we propose a simple, iterative method for building a prescriptive project plan that integrates many project views, including but not limited to control flow, product flow, and measurement:

- Develop the multiple views of the project separately,
- Integrate the multiple views with each other, and
- Initialize target values in the project plan according to existing, local quality models.

Developing multiple views includes representing processes using MVP–L and writing measurement and data interpretation plans using G/Q/M. Integration of measurement views and project plans is achieved by mapping metrics onto project-plan attributes. Iteration between the first two steps will inevitably be required to achieve consistency among the multiple views. Finally, initializing target values in a project plan was illustrated earlier, when the expected resource allocation of the DCT2 project was assigned according to the 30%, 40%, 30% allocation scheme from the re-used project plan.

Further research is needed to define a general theory of consistency between an MVP–L process model and a G/Q/M measurement plan. Two simple types of consistency are defined next:

- A G/Q/M measurement plan and a MVP–L process model are *definitionally consistent* if every G/Q/M metric is mapped to an MVP–L attribute and their respective type definitions are equivalent.
- A G/Q/M measurement plan and a MVP–L process model are *feedback consistent* if all G/Q/M metrics which are intended to guide or manage a project occur in entry and exit criteria of the MVP–L process model.

These basically syntactical definitions of consistency must be supplemented by considerations of the semantics of this integration. We work towards establishing rules for semantic consistency of multiple measurement plans in the context of a single project plan.

5 Using Tractable Project Plans

The primary goal of using tractable project plans is guiding software projects to achieve the needed intellectual control. Providing in-project guidance requires a project plan that meets our definition of tractable; i.e., it is useful for tracking project performance in the course of the project. Guidance may be purely off-line, i.e., in the form of a project handbook, purely on-line, i.e., encoded into a software engineering environment, or somewhere in between the extremes. The major differences along the spectrum between pure off-line and wholly on-line guidance lie in the following issues:

- How can deviations from the project plan be detected?
- How is inter-role communication supported?
- To what extent can actions be performed automatically?
- To what extent can reliable measurement data be collected?
- What quality of guidance can be provided?

This list of differences helps us explain the advantages and required investments involved in using the MVP technology to guide projects. We identify five levels of integrating MVP's process modeling and measurement technology into a conventional software evolution environment and discuss the above list of differences for each level. These levels range from the state of the practice to our ultimate research goals, and are by no means the only possibilities. A conventional environment consisting of software construction tools is assumed as the foundation for all of the following levels:

1. No explicit project plan or measurement plan (predominant state of the practice).
2. Explicit project plan and a documented measurement plan (purely off-line documents).

3a. On-line project plan and a static, off-line measurement plan.

3b. Static, off-line project plan and an on-line measurement plan.

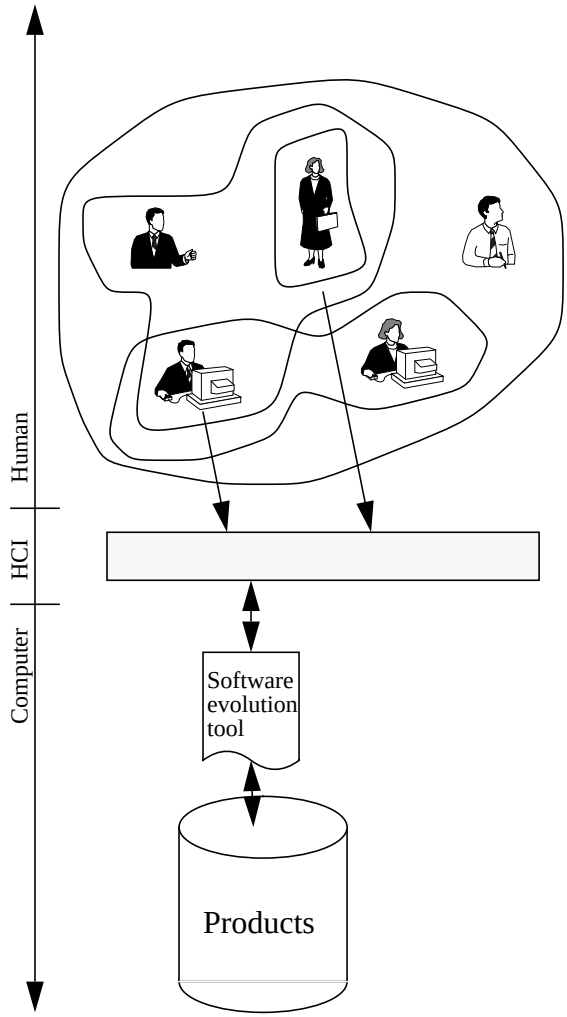

Figure 3: Level 1, No project or measurement plan

4. On-line project plan with an integrated measurement plan
   (our goal, a process-centered environment).

5.1 Level 1: No project plan or measurement plan

This level of technology, as illustrated in Figure 3, reflects state-of-the-practice SEEs and serves as a baseline against which we compare the following levels. Activities performed by individual project members are shown at the top, and actions carried out by a machine are shown below. The interface between project personnel and the software engineering environment is based on the invocation of software tools. A clear understanding of project goals and requirements is necessary for the tools to be used effectively; however, these goals and requirements are only implicitly defined in the minds of the engineers. Because there is no explicit project plan, there is no support in such an environment for detecting plan deviations, supporting inter-role communication, performing actions automatically, collecting reliable measurement data, or providing quality guidance.

5.2 Level 2: Explicit project plan and measurement plan

The next technology increment, as illustrated in Figure 4, reflects current trends in software organizations. Activities are represented using MVP–L project plans, which describe the constructive and analytic process steps. Measurement plans defining project goals are written according to the G/Q/M paradigm, as sketched on the right. Deviations from plan are detected by project personnel by invoking measurement tools and comparing the results thus obtained with the project plan. The project plan may help a team member realize that he or she introduced inconsistency into the project, and that he or she must explicitly notify all affected team members. There is no support for performing actions automatically; all actions must be triggered by explicit user invocations of tools. The measurement plan clearly defines what data is required and when it should be collected, but the physical collection of measurement data requires effort on the project of project personnel to fill out forms. Because such data-collection activities are often postponed or otherwise delayed until the end of the week or month to reduce their overhead, wide variances in data quality may result. Finally, quality guidance may only be provided to the project through the efforts of individuals who track the project state, read the project plan, and monitor the measurement data. There is no guarantee that the project proceeded in accordance with the project plan, which makes static use of the project plan even more difficult.

By using the MVP technology off-line, a project may realize the advantages of improved understanding of their processes, quantitative definition of the criteria necessary for project success, and a clearly defined method by which project state can be monitored. Reaching this level requires an organization to devote considerable effort towards defining its processes and constructing a measurement plan. An example of writing and using explicit project plans, although not using exactly those notations developed in the MVP Project,

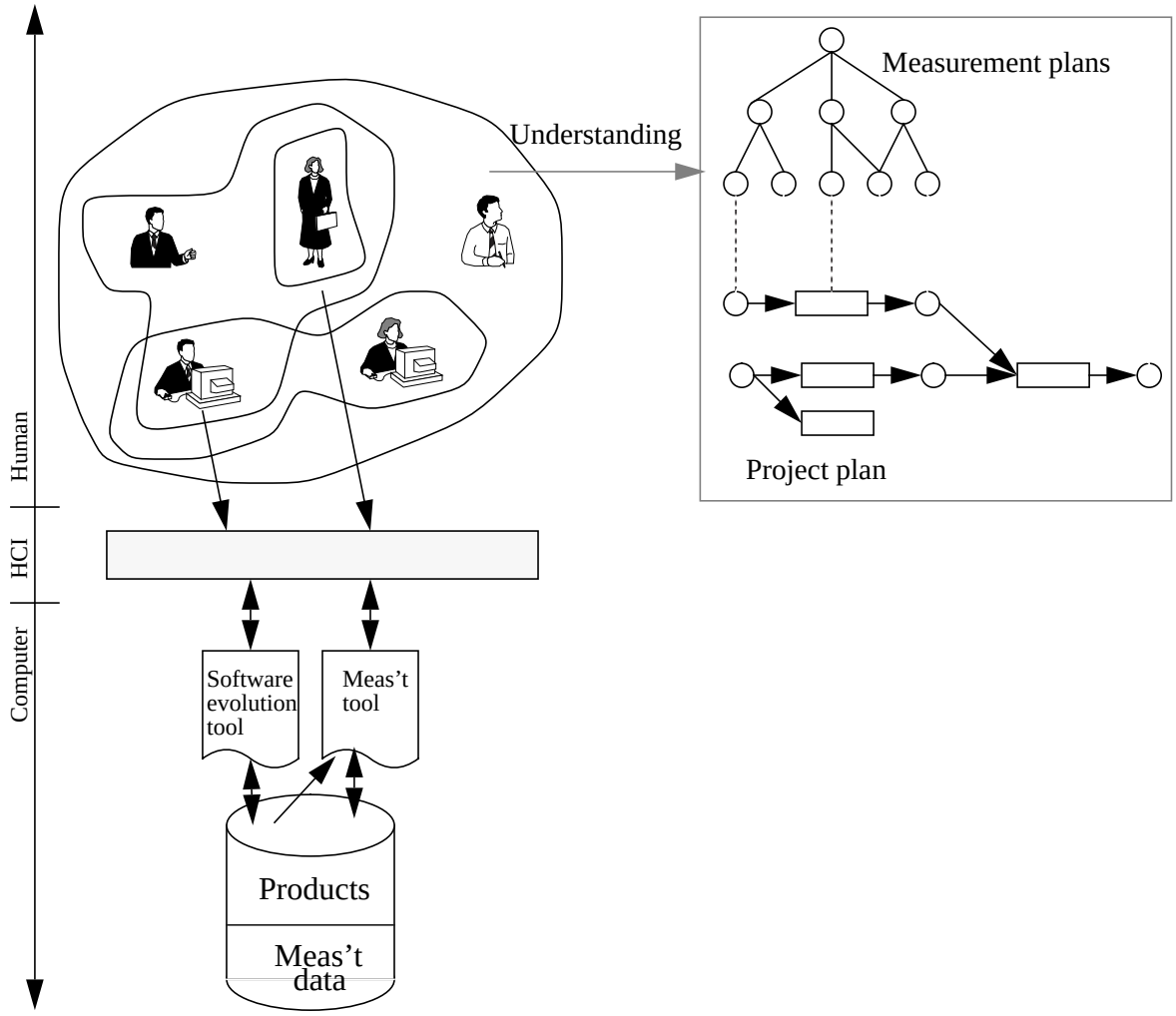

Figure 4: Level 2, Explicit, off-line project plan and measurement plan

is discussed in [28].

Next we describe two real-world examples of having used the MVP technology off-line. NASA's Software Engineering Laboratory used project plans off-line to understand a maintenance process [6, 9]. A first attempt to collect data on maintenance processes in this environment failed because the process model implicit in the minds of the leaders of the study was not consistent with the existing processes. After views of maintenance personnel were iteratively captured, represented using MVP–L, and reviewed until conflicts were resolved, data collection efforts yielded interesting and meaningful results.

The second example is a case study which we performed in cooperation with TRW [8]. TRW personnel used their natural-language description of a proposed reuse-oriented development process to build a project plan using the MVP–L language, analyzed the MVP–L representation for completeness and consistency, changed the representation to fix problems and improve it, and finally rewrote an improved natural-language description. TRW personnel were able to understand MVP–L constructs easily and write meaningful process representations after minimal instruction. The payoff to the users at TRW was an improved understanding of their process and generation of an internally consistent process representation.

5.3 Level 3a: On-line project plan, off-line measurement plan

The possibility of a process-centered environment in which tool use is mediated by a project plan, but without measurement support, is illustrated in Figure 5. Such a system is the first step towards offering project personnel a role-specific, process-centered interface through which they accomplish their work. Because this level of technology offers no on-line support for representing and using the quantitative criteria that define project success, we claim that this level represents no significant progress over use of a sophisticated version control system. The project plan is essentially limited to evaluating process entry and exit criteria in terms of product flow. Deviations from the project plan can be detected when products are changed or not produced as expected. Inter-role communication is provided in the sense that the version control system tracks user's actions and prevents conflicting actions. Actions can be performed automatically exactly when users interact with the revision control system. The collection of reliable measurement data is not supported on-line, so it is subject to the same problems discussed in level 2. Finally, the quality of guidance that can be offered based on product availability is mostly limited to informing project personnel of processes that can begin and notifying processes of changes in input products.

Nonetheless, this technology increment offers many advantages over level 2. Some conflicting actions can be detected automatically, deviations from the plan can be reported, and rudimentary guidance is possible. Further, because we believe that all of these benefits accrue through the use of version control technology, the investment required of an organization to achieve this level is moderate. Version control systems such as RCS [29] are well understood and freely available.

5.4 Level 3b: Off-Line project plan, on-line measurement plan

The possibility of an environment that supports on-line collection and interpretation of measurement data, but does not incorporate a project plan, is depicted in Figure 6. Although the project plan is not automated, it is explicitly defined. Every measurement plan implicitly assumes some process standard. If project performance deviates from that implicit standard, then misinterpretations of the data are inevitable. Any detections of project performance deviations are purely accidental in this environment. In this level of technology, there is no support for inter-role communication. Many trivial activities pertaining to measurement may be automated, especially the routine collection of measurement data. Automated support for data collection can be expected to yield much higher quality data than that which is collected by hand. Finally, limited guidance can be provided if empirical models for the process and environment are available.

The main advantage of this level of technology use is that many trivial activities involved in gathering measurement data can be automated using existing tools. Our hypothesis is that this increased automation will dramatically improve the reliability of the collected data. The immediate return on investment to an organization that achieves this level of integration is higher than at level 3a, primarily because the extra work required by measurement activities can be partially offloaded onto automated tools. As was discussed earlier in this paper, a paradigm like G/Q/M is absolutely neces-

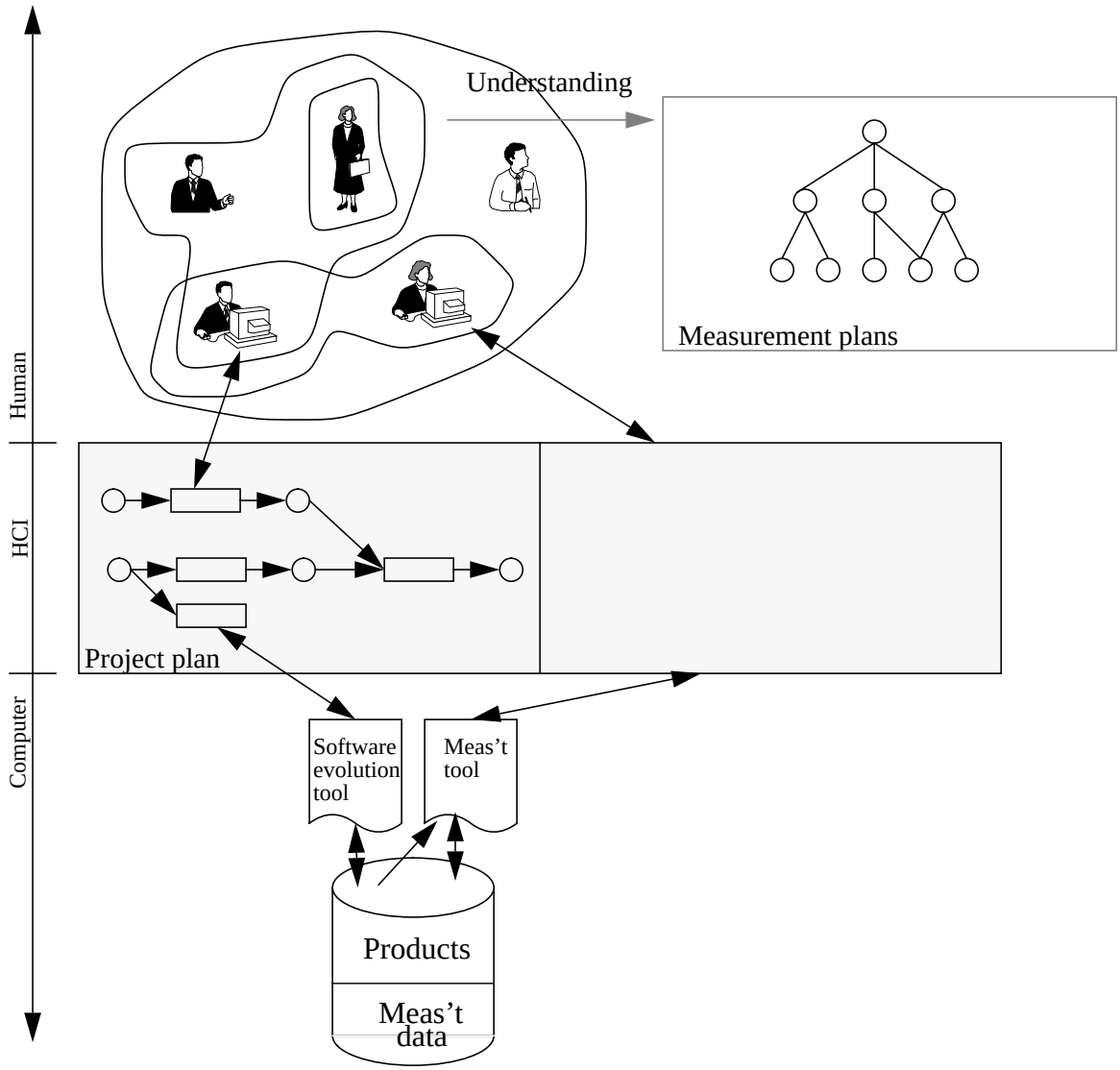

Figure 5: Level 3a, On-Line project plan, off-line measurement plan

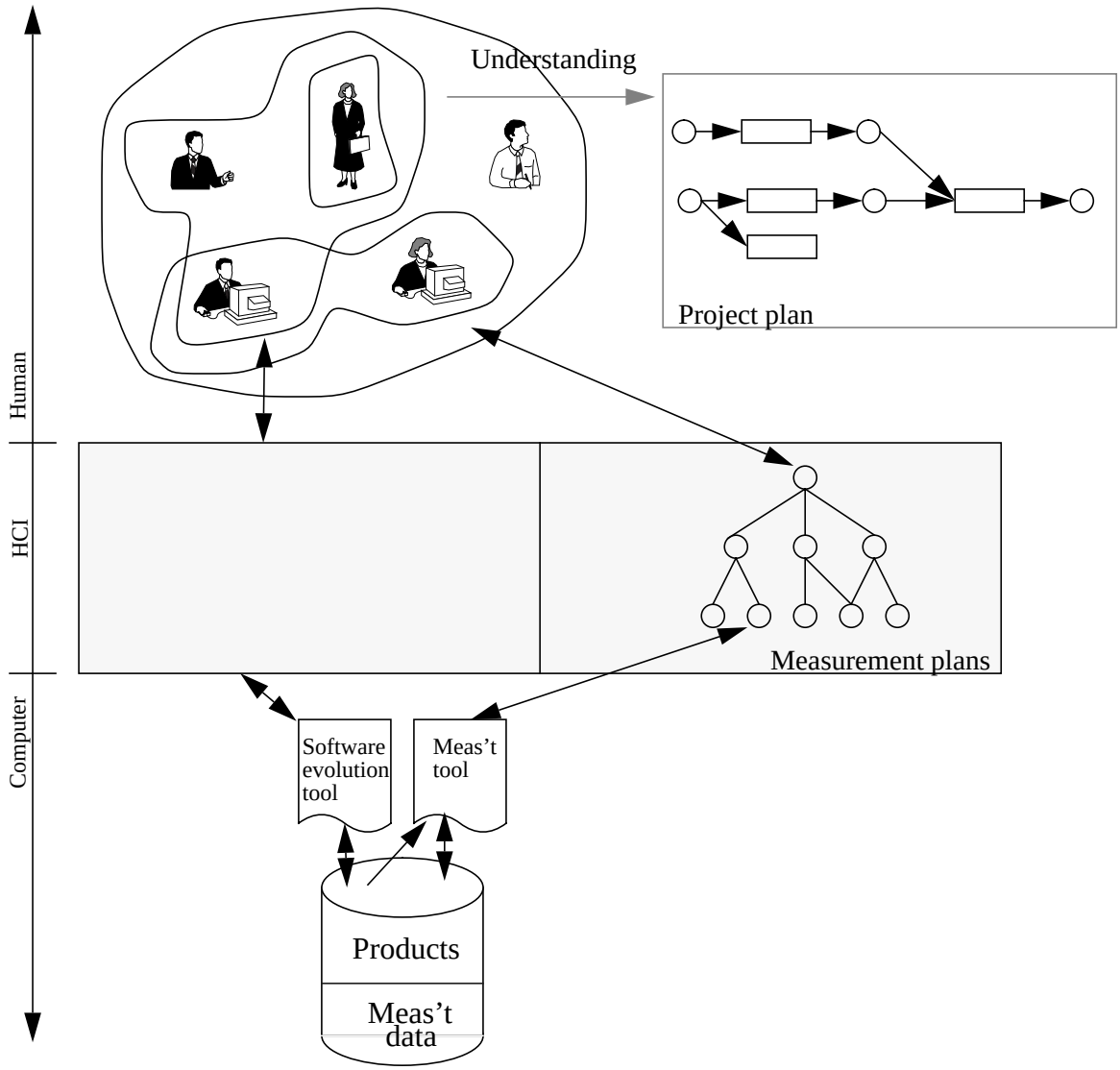

Figure 6: Level 3b, Off-Line project plan, on-line measurement plan

sary to guide the organization to construct a data collection plan, develop baseline models of the organization, and interpret the data.

This level of technology has been implemented in NASA's Software Engineering Laboratory [30, 15]. The Software Management Environment (SME), developed in this organization, offers one example of on-line use of a measurement program [31]. SME accesses large quantities of data captured from the current project and compares that data to models of the typical and target values for each metric in their environment. When current project data deviates from the baselines, the system can offer possible interpretations for the deviations.

5.5 Level 4: Process-centered environment

Our ultimate goal, a system in which quantitative criteria for project success are integrated into an explicit project plan and used to guide a project, is shown in Figure 7. This interface offers indirect access to the usual software evolution tools, and represents a paradigm shift from unmonitored access of software evolution tools to guided, process-oriented access of the same tools. Lines in the figure between the G/Q/M measurement plan and the MVP–L project plan represent the integration of targets and baselines with process models to form a project plan, as discussed in the previous section. The system interface will gather quantitative data and offer guidance according to the collected data and the project plan. By providing such an interface, a project plan serves as an integration mechanism for a software engineering environment [32].

User-initiated evaluations of project state remain possible, but deviations may additionally be detected and reported to project personnel asynchronously; i.e., without explicit user action.

Project personnel are kept informed about which activities are possible, which activities are currently being performed, and which activities have been affected by the unexpected results of a connected activity. Activities as used here are not fine-grained tasks such as compiling a code module, but coarse-grained tasks such as designing a subsystem or conducting a major review. This type of guidance helps teams coordinate their actions among each other and detect when independent actions have come into conflict with each other.

Actions such as periodic data collection or other mechanizable tasks such as compilation may be performed automatically. Such actions may be triggered by changes in project state initiated by project personnel (e.g., starting a process) or by the tracking system (e.g., passage of time).

Data can be collected on a timely basis and an on-line support system can further guarantee the consistency of the project state with the plan. Some types of measurements lend themselves to automatic collection. A simple example is static source-code metrics, which can be collected from a version-control library. Other types of measurements are available only from project personnel. Examples are resources consumed by review activities, characterizations of defects removed, and final completion dates of processes. In all of these cases, automated support of data collection can be provided using something like a forms-based tool to reduce paperwork and encourage timely collection of data. It cannot be overemphasized that automated support for data collection must work harmoniously with project personnel to reduce the costs of this activity, not to check up on personnel.

Finally, given that a deviation from the project plan has been detected, and the project team has not chosen to ignore the deviation, the system can provide facilities to estimate how much work must be redone or to suggest how to replan the project. For example, backward chaining can be used to estimate how far the work must be rolled back in order to restore consistency with the project plan. The ideal process guidance system would warn project personnel well in advance; i.e., before a project state inconsistent with the plan is ever reached.

The capabilities of a process-centered environment can be best understood in the context of an example. We use the DCT1 example to show how a team member responsible for maintaining the consistency of the on-line project plan with the current project state requests information and enters changes in project state.

Assume that the members of the DCT1 project have completed their writing design process and intend to start the writing code process. To keep this example simple, assume that the two processes do not overlap. The responsible team member indicates to the system that the design process is complete. The system responds by collecting information. Data for the design quality aspects of coupling and information hiding is collected using a design-measurement tool, and data regarding resource consumption for the design activity and requirements defects found during the design process is requested from the team member using a forms-based tool. The data are then used to evaluate the exit

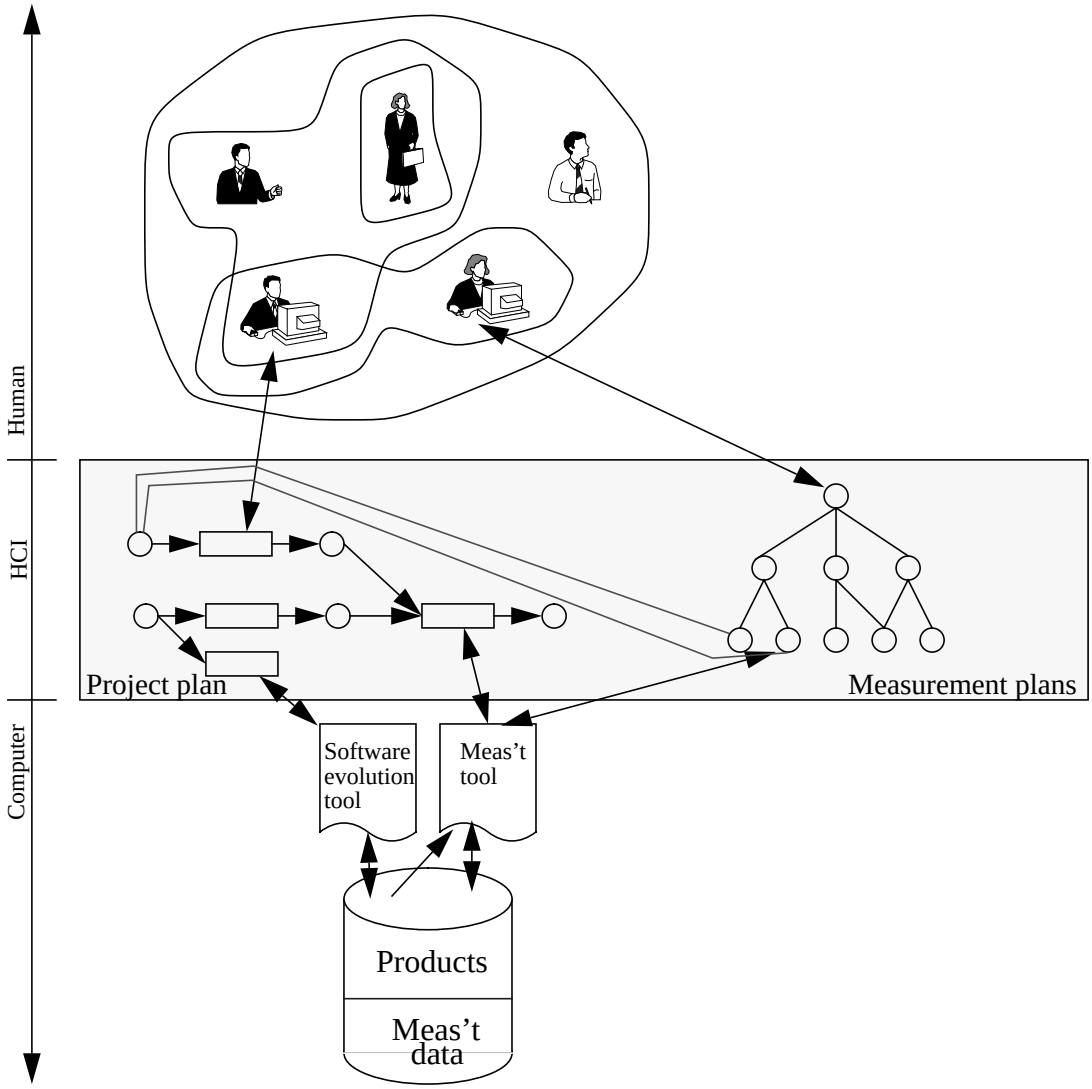

Figure 7: Level 4, A Process-centered environment

criteria for the design process. In this case, assume that the design team has consumed 105% of their resource allocation. This fact is reported. Defect detection data is substantially below the baseline value, and this fact is noted.

Because the exit criteria are not fulfilled due to the excess resource consumption, the user has three options, as discussed earlier. One, ignore the problem; two, repeat the process; or three, replan. Because the resource limit was only exceeded by 5%, the team member decides to ignore the problem and marks the process as complete. The completion of the writing design process fulfills the entry criteria of the writing code process, meaning that it can be started. Because project state is constantly visible to project personnel, this fact is immediately reported. The team member initiates a second change in project state, namely by notifying the tracking system that the team intends to start the writing code process. Because that process's entry criteria are true, this change is accepted. The system's project state is thus made consistent with the fact that team members are proceeding with the writing code process.

5.6 Requirements for Process-Centered SEEs

The discussion of planned capabilities of a process guidance system given above allows us to derive a number of requirements necessary for constructing process-centered software engineering environments. We divide these requirements into two parts, namely those for model building and those for providing in-project guidance. The fundamental concept that we believe must be supported is the systematic definition, collection, use, and evaluation of measurement data.

The modeling machine must contain mechanisms for:

- Defining metrics
- Defining events triggering data collection
- Connecting events with measurement tools
- Analyzing project plans for internal consistency

The execution machine must contain mechanisms for:

- Defining and making project state visible
- Changing project state, such as starting or completing processes
- Querying project state, such as evaluating measurement goals on demand
- Automatically collecting data using tools
- Requesting data from project personnel in a timely manner
- Evaluating all baselines and goals defined for the project automatically to detect deviations
- Maintaining consistency of project state with actual project performance
- Supporting replanning activities in the middle of project performance.

Maintaining the consistency of the project state and supporting replanning are two issues that deserve brief discussion. Project guidance wholly depends on a model of the project that exists only in the support system. Expectations of processes are expressed in the project plan and communicated to personnel. The support system must trust personnel to meet those expectations, and if those expectations cannot be met, then replanning is generally needed.

We understand the term "replanning" to involve changes to a project plan that must be done by people. Following such a replanning process, the altered project plan must be used to restart the guidance system with a project state as close as possible to the one that was current when replanning activities were initiated. Certain types of changes in the project plan, such as deleting subprocesses, will require substantial work to fit the new project plan together with the old project state and allow guidance to continue.

6 Conclusions

Intellectual control over software evolution projects requires the creation of a comprehensive project plan during the planning stage of a project and intelligent support for guiding project performance according to the plan. It is essential to integrate measurable criteria into such plans. We demonstrated in this paper that the currently available process modeling and measurement technologies hold promise for gaining intellectual control over software evolution projects in the context of an improvement-oriented software engineering framework. We suggested one possible integration of process and quality models into measurement-based project plans, both in general and in terms of the technology developed in our MVP Project. Finally, the benefits of explicit measurement-based project plans were

discussed depending on whether such plans are used as off-line reference documents only or whether they are incorporated into a software engineering environment. Our final conclusion is that there is reason to expect qualitative improvements by building so-called process-centered environments. Such environments are based on a change of paradigm regarding the interaction between developers and software engineering environments (from tool-oriented to process-oriented) and promise more intelligent guidance of individual team members and better support for effective software engineering than off-line use of project plans.

References

[1] Frederick P. Brooks, Jr. *The Mythical Man-Month*. Addison Wesley, 1978.

[2] *Proceedings of the International Software Process Workshop*, 1984–1993.

[3] *Proceedings of the International Conference on the Software Process*. IEEE Computer Society Press, 1992–1993.

[4] Marc I. Kellner and H. Dieter Rombach. Session summary: Comparisons of software process descriptions. In Takuya Katayama, editor, *Proceedings of the* 6^{th} *International Software Process Workshop*, pages 7–18. IEEE Press, October 1990.

[5] Mark I. Kellner. Software process modeling: value and experience. In *SEI Technical Review*, pages 23–54. Software Engineering Institute, Pittsburgh, Pennsylvania 15213, 1989.

[6] H. Dieter Rombach and Bradford T. Ulery. Improving software maintenance through measurement. *Proceedings of the IEEE*, 77(4), 1989.

[7] Maria H. Penedo and Christine Shu. Acquiring experience with the modeling and implementation of the project life–cycle process. *IEE Software Engineering Journal*, 6(5):259–274, September 1991.

[8] C. D. Klingler, M. Neviaser, A. Marmor-Squires, C. M. Lott, and H. D. Rombach. A case study in process representation using MVP–L. In *Proceedings of the* 7^{th} *Annual Conference on Computer Assurance (COMPASS 92)*, pages 137–146, June 1992.

[9] H. Dieter Rombach, Bradford T. Ulery, and Jon Valett. Toward full life cycle control: Adding maintenance measurement to the SEL. *Journal of Systems and Software*, 18(2):125–138, May 1992.

[10] Victor R. Basili and H. Dieter Rombach. The TAME project: Towards improvement–oriented software environments. *IEEE Transactions on Software Engineering*, SE-14(6):758–773, June 1988.

[11] Victor R. Basili. Software development: A paradigm for the future. In *Proceedings of the* 13^{th} *Annual International Computer Software and Application Conference (COMPSAC)*, pages 471–485, Orlando, Florida, September 1989.

[12] Frank E. McGarry. Results of 15 years of measurement in the SEL. In *Proceedings of the* 15^{th} *Annual Software Engineering Workshop*. NASA Goddard Space Flight Center, Greenbelt MD 20771, November 1990.

[13] Victor R. Basili and H. Dieter Rombach. Tailoring the software process to project goals and environments. In *Proceedings of the* 9^{th} *International Conference on Software Engineering*, pages 345–357. IEEE, March 1987.

[14] Victor R. Basili and H. Dieter Rombach. Support for comprehensive reuse. *IEE Software Engineering Journal*, 6(5):303–316, September 1991.

[15] National Aeronautics and Space Administration. Manager's handbook for software development. Technical Report SEL-84-101, NASA Goddard Space Flight Center, Greenbelt MD 20771, 1991.

[16] H. Dieter Rombach, Victor R. Basili, and Richard W. Selby, editors. *Experimental Software Engineering Issues: A critical assessment and future directions*. Lecture Notes in Computer Science Nr. 706, Springer-Verlag, 1993.

[17] Karen Huff and Viktor Lesser. A plan–based intelligent assistant that supports the software development process. In Peter Henderson, editor, *Proceedings of the* 3^{rd} *ACM SIGSoft/SIGPLAN Symposium on Practical Software Development Environments*, pages 97–106, November 1988. Appeared as ACM SIGSoft Software Engineering Notes 13(5), November 1988.

[18] Alfred Bröckers, Christopher M. Lott, H. Dieter Rombach, and Martin Verlage. MVP Language Report. Technical Report 229/92, Fachbereich Informatik, Universität Kaiserslautern, 67653 Kaiserslautern, Germany, December 1992.

[19] W. Edwards Deming. *Out of the crisis*. Massachusetts Institute of Technology, Cambridge, Mass., 1986.

[20] Watts S. Humphrey. *Managing the Software Process*. Addison Wesley, Reading, Massachusetts, 1989.

[21] Frank E. McGarry and R. Pajerski. Towards understanding software - 15 years in the SEL. In *Proceedings of the* 15^{th} *Annual Software Engineering Workshop*. NASA Goddard Space Flight Center, Greenbelt MD 20771, November 1990.

[22] J. A. McCall, P. K. Richards, and G. F. Walters. Factors in software quality. Technical Report RADC-TR-77-369, Rome Laboratory, Griffis AFB, NY, 13441, 1977.

[23] M. Kogure and Y. Akao. Quality function deployment and CWQC in Japan. *Quality Progress*, October 1983.

[24] Victor R. Basili, Richard W. Selby, and David H. Hutchens. Experimentation in software engineering. *IEEE Transactions on Software Engineering*, SE-12(7):733–743, July 1986.

[25] H. Dieter Rombach. Practical benefits of goal-oriented measurement. In N. Fenton and B. Littlewood, editors, *Software Reliability and Metrics*. Elsevier Applied Science, London, 1991.

[26] Watts S. Humphrey, David H. Kitson, and Tim C. Kasse. The state of software engineering practice: A preliminary report. In *Proceedings of the* 11^{th} *International Conference on Software Engineering*, pages 277–288. IEEE, May 1989.

[27] ESPRIT. Specific research and technological development programme in the field of information technology. Draft Work Programme for 1993-94, Directorate General XIII, Commission of the European Communities, B-1049 Brussels, Belgium, 1993.

[28] Watts S. Humphrey, Terry R. Snyder, and Ronald R. Willis. Software process improvement at Hughes Aircraft. *IEEE Software*, pages 11–23, July 1991.

[29] Walter Tichy. Rcs–a system for version control. *Software–Practice and Experience*, 15(7):637–654, July 1985.

[30] National Aeronautics and Space Administration. Software engineering laboratory (SEL) relationships, models, and management rules. Technical Report SEL-91-001, NASA Goddard Space Flight Center, Greenbelt MD 20771, February 1991.

[31] W. Decker and Jon Valett. Software management environment (SME) concepts and architecture. Technical Report SEL-89-103, NASA Goddard Space Flight Center, Greenbelt MD 20771, September 1992.

[32] Kurt C. Wallnau and Peter H. Feiler. Tool integration and environment architectures. Technical Report CMU/SEI-91-TR-11, Software Engineering Institute, Carnegie Mellon University, May 1991.