CHAPTER 5

PEOPLE-ORIENTED CAPTURE, DISPLAY, AND USE OF PROCESS INFORMATION

Jens Heidrich, Jürgen Münch, William Riddle, Dieter Rombach

Fraunhofer Institute for Experimental Software Engineering
Sauerwiesen 6, 67661 Kaiserslautern, Germany
E-mail: {heidrich, muench, riddle, rombach}@iese.fraunhofer.de

Project success demands that process performers have accurate, up-to-date information about the activities they should perform, any constraints upon activity performance, and guidance about how to effectively and efficiently perform their activities. The goal of this chapter is to describe support for people-oriented capture, display, and use of process information that experience has shown is highly beneficial. The chapter reviews several state-of-the-art approaches for supporting people-oriented process performance, illustrates challenges of providing this support, and presents experience from practice. We describe different kinds of process knowledge and discuss a method for collecting one kind of process knowledge – measurement data – in a goal-oriented way. We present different ways to display process information in order to satisfy information needs of people involved in a software development project, including the generation of process documentation, role-based workspaces, and control centers for software development. Furthermore, we illustrate how process information can be used to support process performance through the use of not only workspaces and control centers but also process enactment and experience management. The approaches presented in this chapter can be seen as a contribution towards supporting people-oriented software development.

1 Introduction

Well-designed, accurately-performed processes are critical to the successful conduct of an organization's projects. This is particularly true

Reference:
Jens Heidrich, Jürgen Münch, William Riddle, Dieter Rombach, "People-oriented Capture, Display, and Use of Process Information", In: New Trends in Software Pro-cess Modeling, (S. T. Acuña and M. I. Sánchez-Segura, eds.), Series on Software Engi-neering and Knowledge Engineering, Vol. 18, World Scientific Publishing Company, pp. 121-179, 2006.
DOI: 10.1142/9789812774460_0005
http://www.worldscientific.com/worldscibooks/10.1142/5995

when developing large systems by carrying out many, highly interdependent, activities. Some activities are quite simple but others, such as project planning, coordination, cooperation, control, and improvement, may be quite complex. Some activities may be "algorithmic" (defined by concrete procedures) and some may be "creative" (stochastically undetermined). Some may be performed by teams of developers; others may be enacted by machines. All activities must contribute to meeting the project goals which will vary across projects.

As a result, the processes must be designed for effective, efficient, and accurate performance in a wide variety of contexts. Three particularly difficult challenges are:

- The conduct of development processes cannot be completely automated because performance is largely human-based and, therefore, has to deal with non-deterministic behavior of human process performers. This often causes problems such as non-predictable deviations or confusion regarding roles and responsibilities.
- Contexts characterize a project's environment and consist of organizational, technical, and personal characteristics that influence the project's processes and their performance. The relationship of the context and the processes is often quite hard to understand, especially for the development of large-scale, complex systems. This makes it difficult for process performers to obtain the necessary or appropriate information and assess the impacts of different contexts.
- The contexts vary between projects and even within projects. Because development activities are context-dependent, the processes need to be adapted to different contexts and this adaptation is often quite difficult. In addition, the processes may need to be adapted "on the fly" during project performance. Rapid context switches often lead to unnecessary rework when context switches are insufficiently supported.

The development of large and complex systems that may include hundreds of hardware and software components is quite complex because of these challenges. Other complexities occur in small, team-based development projects. Many problems are related to the fact that software development is largely human-based. People need help in meeting these challenges. This includes customizing process information to specific needs for different contexts. Necessary process support includes improved communication, detailed reasoning about process features, guiding people when performing processes, improving both processes themselves and their results, and automating process steps to gain deterministic process behavior[1, 2]. People-oriented process support requires managing large and complex processes contexts by reducing the cognitive load, supporting long-living processes by providing mechanisms for context switching, and supporting collaboration by applying team-based and multi-disciplinary approaches.

The goal of this chapter is to describe support for people-oriented capture, display and use of process information that experience has shown is highly beneficial. The chapter reviews several state-of-the-art approaches for supporting people-oriented process performance, illustrates the challenges of providing this support, and presents experience from practice. Most of the approaches and examples stem from the software engineering domain; i.e., the processes are "software development processes". However, much of the material in this chapter can be applied to "development processes" from other domains (such as development processes for mechanical systems) or other – business-oriented – process domains within an organization, such as marketing and sales processes.

1.1 General Concepts

People-oriented process support requires mechanisms supporting the interface between processes and people; this is, in essence, the focus of this chapter. The concepts of *role* and *agent* are fundamental to discussing these mechanisms and are therefore explained in this sub-section.

A role definition indicates the role's purpose with respect to the process. In essence, role definitions define the parts that people play as they participate in carrying out the process. Role definitions are analogous to the part definitions found in a script for a theatrical production. An important aspect is that the role definition is specific to the process. If role definitions in different process descriptions have the same name, then this is incidental. The purpose, scope, and nature of the role are solely as specified in its definition for the specific process.

Just as people play the parts in a theatrical production, people play roles in process performance. To specify that a person is playing a role, we say the person *occupies* the role. Further, we do not talk in terms of specific people (Bob, Michele, or Pete), but rather talk in terms of agents.

The net effect is that there are two levels of abstraction reflecting two very useful separations of concern. The role vs. agent abstraction allows the separation of a concern for what a person must do when participating in a particular process from a concern for what the person must be able to do as an employee who may be assigned to participate in various processes. This separation of concern is highlighted by noting the difference between a role definition, which is process specific, and a job description, which is position specific and usually fairly neutral about the specific processes the employee will be assigned to.

The agent vs. person abstraction allows a separation of concern for a general spectrum of qualifications, abilities, and experiences — as typically indicated in a job description — from a person's specific qualifications, abilities, and experiences. This abstraction has two important side-effects. First, it allows a group of people to share a particular position. An example is a `System Administrator` position which is often filled by assigning a group of people who each work part-time in this position. The other side-effect is that it becomes possible to think of filling a particular position with a tool, effectively automating the position. An example is a `Request Filter` agent that filters incoming requests into several different "bins," a capability that is necessary for a variety of processes and may quite often be automated.

A role definition must treat three different aspects of the role as part of a process:

- Responsibilities: the role's obligations and permissions with respect to the process; for example `create financial report`, `assure project success`, and `can access employee records`.
- Activities: the role's participation in activities, perhaps accompanied by time sequencing information; for example, `gather the requirements` and `develop the design`.

- Abilities: skill and experience requirements for agents who occupy the role; for example, `trained in using Word` and `familiar with the Delphi approach to brainstorming`[a].

These three aspects of a role correspond to three different approaches to supporting people during process performance. All three approaches allow agents to find and focus on an activity they must carry out; the three approaches vary with respect to the view presented to the agent as a starting point. In a responsibility-based approach the agent starts with a view that emphasizes its responsibilities. In an activity-based approach, however, the starting point reflects a collection of inter-related activities and the agent selects one of the activities that the role participates in. Finally, in an ability-based approach, agents start with a view indicating the capabilities needed to successfully occupy a role and the agent may select an activity that requires a specific capability. In this paper, we focus on responsibility and activity-based approaches.

People-oriented process support needs to consider the human characteristics of individuals and teams. These characteristics comprise motivation (i.e., the stimulus a person has about achieving some result), satisfaction (i.e., the fulfillment of a need or the source or means of enjoyment), skills (i.e., knowledge and training), and experience (i.e., practical knowledge and experience resulting from observing or participating in a particular activity).

Process support should be customized to human characteristics and it should recognize that it can influence these characteristics. The motivation of a highly experienced developer, for example, could be decreased by prescribing finely detailed approaches to carrying out activities. The skills of an inexperienced developer could be amplified by providing detailed guidance for an activity.

Human characteristics may have a major influence on the results of a development project. The following empirical findings and hypotheses

[a] These statements could appear in a job description to specify criteria for evaluating potential employees. Here, they are being used to specify the criteria for some agent assigned to the role. These are analogous uses, but the first concerns an agent's general skills and experience whereas the second concerns the skills and experience needed for a specific process.

(discussed in "A Handbook of Software and Systems Engineering"[3]) are related to skill, motivation, and satisfaction and can be applied for designing people-oriented process support. According to Kupfmüller's law, humans receive most information visually. It is important to consider that not all senses are involved equally during the reception of process information. The predominance of the visual system could be exploited for displaying process information via pictures, graphs, or signs. Krause's law states that multimodal information is easier to remember than single-mode information. Including multimedia capabilities in process support could utilize this finding. Miller's law, defined by the psychologist George Miller, says that short-term memory is limited to 7 +/- 2 chunks of information. This indicates how much the complexity of processes and contexts needs to be reduced for displaying them adequately to process performers. Many other empirical findings should be considered when developing people-oriented process support, for example: human needs and desires are strictly prioritized (Maslow-Herzberg's law); motivation requires integration and participation (McGregor's hypothesis); and group behavior depends on the level of attention (Hawthorn effect).

These empirical findings and hypotheses should be carefully considered when providing people-oriented process support because adherence or non-adherence to them is a major determiner of success or failure.

1.2 Contents

The chapter is organized into three parts regarding capturing process information, its display, and its use. Section 2 addresses the collection of different kinds of process knowledge and discusses the Goal Question Metric (GQM) paradigm as a goal-oriented method for collecting process-related measurement data. Section 3 deals with the different ways process information may be displayed in order to satisfy process performer information needs. In the section, we discuss the generation of process documentation, role-based workspaces, and control centers for software development. Section 4 illustrates how process information can be used during process performance. In the section, we discuss the importance of not only workspaces and control centers but also process enactment support. In the section, we also discuss experience management issues; that is, various ways in which process information can be used to improve future process performance. In each Section, we discuss related work and future trends as appropriate. Finally, section 5 gives a brief summary of the material in this chapter.

Fig. 1 gives an overview of the sections in terms of their relationship to a simple development process and its major roles.

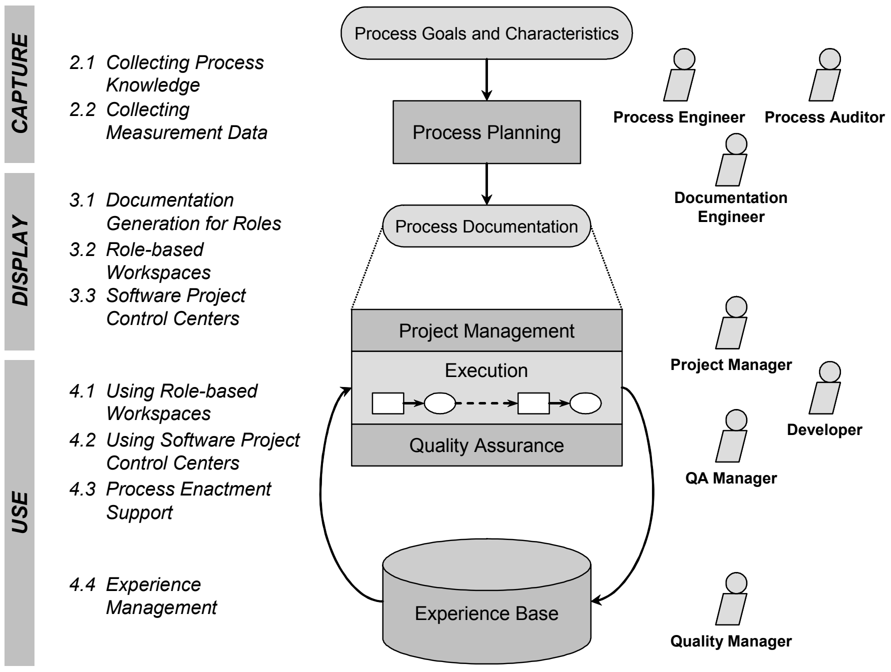

CAPTURE
2.1 Collecting Process Knowledge
2.2 Collecting Measurement Data
DISPLAY
3.1 Documentation Generation for Roles
3.2 Role-based Workspaces
3.3 Software Project Control Centers
USE
4.1 Using Role-based Workspaces
4.2 Using Software Project Control Centers
4.3 Process Enactment Support
4.4 Experience Management
Process Goals and Characteristics
Process Planning
Process Documentation
Project Management
Execution
Quality Assurance
Experience Base
Process Engineer
Process Auditor
Documentation Engineer
Project Manager
Developer
QA Manager
Quality Manager

Fig. 1. Section Focus with respect to a Simple Development Model (not addressing all Planning and Enactment Issues).

2 Capturing Process Information

Access to information about a process is critical to its effective, efficient and accurate performance. This information comes from many sources: process designers, process performers, customers and other stakeholders concerned with the outcome, and corporate executives interested in the degree to which the processes support achieving the organization's business objectives. Once collected, the information is typically organized into process handbooks, with different handbooks rendering the information as needed for particular roles (for example, a `Developer Handbook` and a `Process Auditor Handbook`).

In this section, we the first define various kinds of process information. After briefly discussing the collection of these kinds of process knowledge, we then address the collection of measurement data (one kind of process knowledge) in more detail.

2.1 Collecting Process Knowledge

There are many different, but inter-related, kinds of process-related information:

- Process Assets: templates, checklists, examples, best-practice descriptions, policy documents, standards definitions, lessons-learned, etc., useful during process performance.
- Process Definition: information about a specific process including: the activities to be performed, the roles that agents occupy when performing the activities, the artifacts used and produced during activity performance, the conditions that reflect progress, and the assets pertinent to the process.
- Process Status: information useful for controlling or tracing process performance in terms of activity sequencing, role/agent assignments, the degree of artifact completion, the usage of assets, etc.
- Process Measurement Data: information characterizing a particular performance of a process in terms of the expended effort, performance of the personnel, resource utilization, etc.
- Project-specific Data: information created during a process performance including: specific documents created and worked on during the project, records of group meetings, white papers discussing various design options, lessons learned, etc.

In the following sections we discuss the collection of each of these kinds of process knowledge.

2.1.1 Process Assets

Process assets may be collected from several sources. Standardization efforts[4] not only define the standard but also provide example templates and checklists as well as examples of good (and bad) uses of these assets. Definitions of Process Maturity Frameworks, for example the CMM[5], are also often the source of templates, checklists, and examples of good or bad artifacts. Framework-oriented sets of assets are also commercially available (for example from pragma[6] and the FAA[7]).

Corporate-wide process definition groups are normally responsible for developing assets supporting the organization's processes. This primarily includes templates, examples and checklists. It also includes the definition of best practices as they should be practiced within the organization.

These process definition groups are also frequently responsible for assuring that the organization's process assets are influenced by the experience gained during project performance. This includes collecting new templates, checklists, examples, etc. It also includes collecting lessons learned. Finally, it includes updating existing assets to reflect experience in using them.

Many organizations have devoted considerable effort to collecting their assets, and making them readily available, in a corporate-wide Process Asset Library (PAL). A PAL organizes the assets according to various cataloguing dimensions. These dimensions reflect the assets' general characteristics (their general types, the application programs needed to use them, reactions stemming from their use, their pertinence to various types of projects, etc.). An asset's pertinence to specific processes may also be used as a cataloguing dimension. In the following, however, we indicate that process definitions may provide better, role-specific as well as process-specific, "doorways" into a PAL.

2.1.2 Process Definition

Information about a process may be collected in several ways. Typical alternatives are observing real projects, describing intended activities, studying the literature and industry reports, and interviewing people involved in a project. These approaches to gathering process knowledge are varied and may be combined in many ways. All of the various ways result, however, in one of two distinctly different kinds of process definitions: *descriptive process models* and *prescriptive process models*.

Descriptive process models describe processes as they take place in real projects, i.e., descriptive process models are the result of observation. Descriptive models describe how a particular software system was developed, are specific to the actual process used to develop the system, and may be generalized only through systematic comparative analysis[8]. They can be used for analysis purposes and are often the basis for understanding and improving existing practices. Descriptive process models can be used as a baseline for improvement programs and they can be a basis for identifying strength and weaknesses. Descriptive process modeling is a good means for obtaining a process description that accurately describes the real process. This helps gain a deeper understanding of the process, a prerequisite for effective process measurement and assessment.

For conducting descriptive modeling, Becker, Hamann, and Verlage[9] propose a two-phase procedure consisting of eight steps. The set-up phase comprises the configuration of the process modeling approach. The steps in this phase are performed relatively infrequently. The execution phase concerns a particular use of the process. All steps in the execution phase should be performed for each inspection of a particular use. An enumeration of the phases and steps is:

- Phase I: “Set-up”
  - Step 1: State objectives and scope.
  - Step 2: Select or develop a process modeling schema.
  - Step 3: Select (a set of) process modeling formalisms.
  - Step 4: Select or tailor tools.
- Phase II: “Execution”
  - Step 5: Elicit process knowledge.
  - Step 6: Create model.
  - Step 7: Analyze the process model.

o Step 8: Analyze the process.

One approach to descriptive modeling, regardless of how it is conducted, is *multi-view modeling* (MVM)[10]. In this approach, the underlying idea is to reduce process description complexity by eliciting role-oriented process information (e.g., from the point of view of a `Tester` or a `Requirements Analyst`) because people are unable to look at complex processes as a whole and usually focus on information relevant only to their roles. Because roles collaborate, there is typically an overlap of process information in different role-oriented views. Experience has shown that different people occupying different roles in a project often have a varying understanding of the overall process. MVM permits information from different role-oriented views to be integrated to develop a consistent, comprehensive descriptive process model.

MVM supports the elicitation of role-oriented process models, the analysis of similarities and differences between these models, the analysis of consistency between different models, and finally the integration of various views into a comprehensive process model. Benefits of this approach are: it considers the perspective and experience of individual persons or teams, it supports role-oriented modularization of complex process models, it supports the identification of inconsistencies and inefficiencies between role-specific process views, and it allows the reuse of role-specific views.

In contrast to a descriptive process model, a prescriptive process model defines guidelines, frameworks and details for a new approach to carrying out the process (Scacchi[8]). It specifies how activities should be performed, and in what order. It also establishes conditions that can be used to guide or track process performance; this is discussed in the very next section. Finally, it establishes contexts in which process performers may effectively, efficiently and accurately satisfy the responsibilities for the roles that they occupy; this is discussed in section 3.2.

2.1.3 Process Status

Process status information characterizes the state of process performance. The fundamental notion is *process element state*: a measurable attribute of some process element. For example, an artifact's state might be `drafted`, `approved`, or `rejected`. As another example, the state of an activity might be `suspended`, `active`, or `completed`. A statement about a process state that can be used to control the process or characterize progress is called a *condition*: a Boolean expression referring to the states of process elements. An example condition is `Design Document approved and Coding Standards identified`. The validity of a condition changes over time as a result of *events*: actions that establish new states for process elements (or perhaps merely re-establish their existing states). An example event is `design review finishes` which might establish the state of a `Design Document` as either `approved` or `rejected` and thereby affect the validity of conditions defined with respect to the state of the `Design Document`.

Status information may be collected by the techniques identified in previous sections (for example, by interviewing process performers). The most important purpose is to collect information about the possible states for process elements (for example, the possible states for a `Financial Report` artifact). This is often simplified by identifying states for types of process elements, for example, by indicating that all artifacts may have the states `drafted`, `approved`, or `rejected`. Another purpose of status information collection is to identify the conditions needed to assess progress and activity pre- and post-conditions useful for controlling process performance. A third purpose is to identify the events that change process element states and affect the validity of conditions; the events are normally strongly tied to points in the definition of an activity at which the events occur.

Once collected and used as part of a prescriptive process definition, status information may guide the collection of data about the actual status at various points during process performance in order to check that the actual performance matches the intended performance. Collection of actual status information during process performance requires instrumentation of the performance-time support system. Amadeus[11] is an example of a system providing this instrumentation; it uses scripts to control data collection and these scripts can be developed using the status information in a prescriptive process description.

The status information in a prescriptive process definition may be used to control, rather than merely track, process performance. This use of status information to support process enactment is discussed is section 4.3.

2.1.4 Process Measurement Data

Process measurement data are collected during process performance and characterize special aspects of the process, related products, and resources, especially quality-related aspects. Example quality-related aspects include: the effort in person-hours a certain developer spent on a certain part of the process, the performance in lines of code per hour of the coding personnel, or tool usage in hours. Process measurement data provides quantitative information about process performance to complement the qualitative status information.

While project status information is collected in order to track or control the state of the overall project, measurement data are usually collected in order to control certain quality aspects. A project manager would basically be interested in the former and a quality assurance manager in the latter. Therefore, status information and measurement data provide qualitatively different, but complementary, views of process performance.

To control a software development project, it is crucial to know what measurement data have to be collected relative to defined measurement goals of the project. We address this issue in more detail in section 2.2.

2.1.5 Project-specific Data

Project-specific data are created by carrying out the process. For example, performing a `Create Design` task will create a `Design Document` that is a specific instance of the `Design Document` artifact specified in the process description. Project-specific data are collected during process performance as the agents occupying the process' various roles create and modify the specific artifacts produced during process performance.

Project-specific data provide a concrete record of the effects of a process performance, complementing the status information and measurement data characterizing how these effects were obtained. Most importantly, this includes the intended results of the process. It also includes other documents such as records of project meetings and white papers discussing various design options.

2.2 Collecting Measurement Data

Measurement data are needed to analyze quality aspects of a project, the software development process used by the project, the products it produces, and the resources it uses[12]. Examples of measurement data are: the effort of the process `Create Requirements` in person-hours, the complexity of the product `Requirements Document` in function points, or the cost conformance of the project `Building Automation System` in US dollars above or below the planned costs.

But how can we decide what to measure and how to interpret the collected measurement data? Basically, we may distinguish two types of measurement approaches. The first one starts with measurable observations and relates them to measurement objectives and goals. We call this approach bottom-up because it starts with concrete measures and ends up with abstract measurement goals. The second type starts with the definition of a measurement goal and derives concrete measures from it. We call this approach top-down because every measure is derived and interpreted in the context of a specific goal. Starting with a measurement goal eases the development of adequate, consistent, and complete measurement plans.

One top-down approach is the Goal Question Metric (GQM) paradigm developed by Victor Basili and David Weiss[13]. Its main idea is to define a measurement goal and systematically derive questions and finally metrics. The approach can be applied to software processes, products, and resources and gives guidance on how to derive metrics in a goal-oriented way.

Before describing this approach in more detail, it is important to define some basic concepts:

- A *software entity* is a specific process, product, or resource pertaining to a software development project for which measurement data are being collected. For example, `Create Requirements`, `Create Design`, `Coding`, and `Testing` are process entities.

- An *attribute* is a characteristic of a specific entity. For example, the length of the product `Code` is a product-related attribute.
- A *metric* is a (numerical or enumerated) scale that characterizes an attribute of a specific entity. For example, the length of product `Code` can be measured as lines of code (including or excluding comments).
- *Measurement* is the process of assigning certain values (defined by the metrics) to attributes of software entities.
- *Measurement data* is the set of concrete data for all metrics measured during the performance of a software development project.

Some metrics can be measured directly, whereas others have to be computed from other metrics. We call the first *direct* and the latter *indirect metrics*. For example, the indirect metric `simple design complexity` for an object-oriented system may be computed from the direct metrics `number of classes` and the `number of relationships among classes`.

2.2.1 Definition of GQM Plans

The first step in setting up a measurement plan using the GQM paradigm is to define all of the goals for a measurement program. Each GQM goal definition consists of five different components[14]:

- The *object* defines the central node for measurement, namely the object we want to analyze. For example, a certain process, product, or resource, or even the overall project may be a legal measurement object.
- The *purpose* describes the intention we have in setting up a measurement plan. For example, we want to characterize, improve, control, predict, or simply analyze the measurement object.
- The *quality focus* defines the characteristics of the analyzed object we are interested in. For example, reliability, usability, security, safety, scalability, performance, efficiency, or maintainability of the measurement object.

- The *viewpoint* describes the perspective from which a quality aspect is analyzed. For example, a developer, the project manager, the quality assurance manager, or the customer. This component is especially important to supporting people-oriented measurement; it helps in identifying the different groups of people interested in the collected measurement data and in avoiding needless data collection.
- The *context* defines the environment where measurement takes place. Usually, measurement data highly depend upon the context they are collected in, and the results that are obtained are not transferable to different environments. For example, effort data originating from a development project for embedded systems is probably not transferable to web-based application development.

An example of a measurement goal definition is: Analyze *the inspection process* for the purpose of *improving* with respect to *efficiency* from the viewpoint of *the quality assurance manager* in the context of *company A's implementing the automation system BAS 2004*.

The second step in setting up a GQM-based measurement plan is to derive questions for each measurement goal. Answering these questions should support assessing the measurement goal. We distinguish between two types of questions: Questions regarding the quality focus of the measurement goal and questions regarding variation factors, i.e., factors that influence the values of the quality focus measurements. The latter type of question can further be divided into questions regarding processes (for example, process conformance, domain understanding, and so on) and questions regarding products (for example, logical and physical attributes of products such as size and complexity, development costs, and changes). Questions illustrating a quality focus are:

- How many defects are in the requirements document?
- What is the distribution of the defects with respect to a set of defect classes?
- How many defects were found when inspecting the requirements document?
- How much does it cost to fix all the defects found in the requirement document?

On the other hand, questions that influence the quality focus measurements and therefore address variation factors are:

- Process-related: How experienced are the developers? What kind of inspection technique is used?
- Product-related: What is the complexity of the requirements document?

If a question is too complex to be answered directly, we may refine it to a set of simpler, more specific, questions. For example, we can refine the question regarding requirements document complexity to the following three, very specific, questions: How many functional, non-functional, and inverse requirements and design decisions are listed? How many use cases are included? The composition of the answers to the more specific questions leads to answering the original, complex, question.

After defining all of the measurement goal-related questions (and refining them), we are able to derive some metrics that help in answering the questions. This is the third step of the GQM approach. We can define more than one metric for a single GQM question. For example, regarding the question concerning developer experience, we can measure the number of years employed for each developer, the general degree of experience (e.g., `high`, `medium`, `low`), or the developer's experience with a certain requirements analysis tool. As mentioned, each metric has values along a certain scale and the scale indicates which values may be combined, and in which ways, to get valid results. For example, we may add the rework effort for every developer of the requirements document (because the values are on a rational scale), while values of the general degree of developer experience may not be added (because they belong to an ordinal scale and the result of adding `high` and `low` is not defined by the scale).

An important question is, "How can we find a set of relevant measurement goals and derive questions and metrics?" An enhancement of GQM uses *abstraction sheets*[15] to identify components of measurement goals and to analyze the quality focus and its variation factors in more detail. In essence, an abstraction sheet summarizes a GQM plan. An example is shown in Fig. 2.

Object	Purpose	Quality Focus	Viewpoint	Context
Requirements Inspection Process	Characterize	Efficiency	Quality Assurance Manager	company A's building automation system BAS 2004

Quality Focus	Variation Factors
QF1: Number of included defects	VF1: Experience of developers
QF2: Distribution according to defect classes	VF2: Inspection type
QF3: Percentage found during inspection process	
QF4: Total rework effort in hours	

Baseline Hypotheses	Impact of Variation Factors
QF1: 100	VF1 high => QF1 low => QF4 low
QF2: 40% omission, 30% ambiguous information, 20% incorrect fact, 10% miscellaneous	VF2 perspective-based => QF3 high
QF3: 20%	
QF4: 1000 h	

Fig. 2. Sample of a GQM Abstraction Sheet.

As shown in the figure, an abstraction sheet consists of five different sections. The first section (at the top of the example sheet) gives an overview of all the components of the measurement goal. The second section identifies direct metrics related to the quality focus. Not all metrics defined in a GQM plan are represented on abstraction sheets, only the most important ones. The third section identifies variation factors; that is, metrics that influence the values of quality focus metrics. (These two sections appear in the center of the example sheet.) The fourth section defines predicted values for the metrics defined in the quality focus section; that is, it identifies expected values hypothesized before collecting the real measurement data. These data can be used to control the measured quality aspects of the project during project execution. The fifth section indicates the expected impact of variation factors for the metrics identified in the quality focus section. (These two sections appear at the bottom of the example sheet.) An abstraction sheet is usually constructed during structured interview sessions with all of the measurement program's stakeholders. The GQM viewpoint helps in identifying the interests of different project (and organization) roles and in inter-relating different measurement needs.

2.2.2 Application of Measurement Plans

After defining the measurement plan (e.g., a GQM plan), the next step is to select techniques and methods for collecting the measurement data. This includes assigning persons responsible for data collection and validation. The tasks that are necessary to perform data collection are called *data collection procedure*s. Van Solingen and Berghout[12] suggest that at least the following questions must be addressed when setting up data collection procedures:

- Which person should collect which metric?
- When should a person collect a metric?
- How can measurement data be collected efficiently and effectively?
- To whom should the collected data be delivered?

We can distinguish between data collected automatically by a tool, such as lines of code or design complexity, and data that have to be collected from people, such as effort in person-hours for a certain process. Experience has shown that, in general, the most valuable information is that collected from people rather than by tool-based analysis.

Data can be collected from people through manual (paper-based) forms or electronically (e.g., via web-based forms, e-mail, or spreadsheet forms). The greatest advantage of an electronic data collection system is that the collected data may be used for project control purposes (if interpretation and analysis of the raw measurement data can be automated).

Data can be collected when special events occur (such as when a project milestone is reached, at the end of major activities, or upon completion of important products) or continuously at periodic time points (e.g., every day or once a week). The best collection strategy to use for a metric depends on several factors, such as the collection approach (automatic versus manual data collection), the metric being measured, and the people entering the measurement data.

3 Displaying Process Information

The information needs of the people involved in a software development project varies according to the roles they occupy. A `Project Manager`, for example, needs different process information than a `Developer`. The former will be interested in the state of the overall project (e.g., a list of all uncompleted tasks), while the latter is most interested in process information regarding specific development activities (e.g., how a certain activity has to be performed). The goal of this section is to illustrate a variety of ways process information can be presented and how different project roles may benefit from different presentations. First, we describe the generation of process documentation in order to support different project roles. Secondly, we discuss role-based workspaces displaying process information in an organized, role-specific, manner. Finally, we describe control centers for software development as a systematic way for interpreting and visualizing measurement data.

3.1 Documentation Generation for Roles

Process documentation is generally thought of as information that helps process performers do the "right thing" at the "right time." As such it defines the work that should be done, dependencies among work packages assigned by the project manager, the responsibilities of the various roles, the conditions that control the sequencing of the work, the conditions that can be used to track progress, etc. Process documentation is most usually organized according to a logical decomposition of the work that must be done with chapters for major phases, sections for major activities within each phase, and sub-sections for specific tasks.

Process documentation content and organization reflect task-related questions posed by *project members*, agents who are assigned to a project and responsible for carrying out the process. Typical questions are: What do I have to do to perform a task? Which tasks am I responsible for? How can I determine whether or not I have successfully completed a task? Where can I find a particular template?

Across all the roles in a process there will be many questions of many different kinds. For each kind of question, there will be one or more views that help in answering questions of this kind. For example, an activity decomposition view helps in asking questions about the interdependencies among activities and an artifact lifecycle view helps in answering questions about artifact state sequences.[b]

The information needed to answer some role's questions is typically documented in handbooks. The use of handbooks for software development processes has been recognized widely as beneficial in order to perform systematic, traceable projects. Nevertheless, software developers generally face problems in using the handbooks typically provided for software development processes. The reasons for these problems include:

- The handbooks are lengthy, perhaps hundreds of pages, and often not very well structured, thus complicating information retrieval.
- The handbooks are frequently out-of-date. Because new versions are difficult and time-consuming to produce, they are infrequently developed.
- Formal notations may be used to achieve high degrees of precision; graphical representations may used to increase the understandability of descriptions using these notations. Process descriptions are, however, usually informal. Among other things, this makes it hard to customize the processes to match the characteristics and needs of a specific project.
- The dynamic behavior of the process is not well-specified, again because the descriptions are informal. Ambiguity is common, and different agents have different understandings of the behavior.
- The consistency, clarity, and completeness of informal software process descriptions cannot be easily ensured. Costly, lengthy reviews are therefore needed to assure consistent, clear, and complete handbooks. These reviews are frequently not done.

Even more problematic is that <u>one</u> handbook cannot conveniently meet the needs of all the roles having questions about the process and its

[b] In paper-based documentation, one view is, of necessity, used in the body of the documentation and other views are provided in appendices. In web-based documentation, all of the various views may be provided as "top-level doorways" into the process information.

performance. Many handbooks are needed, each oriented towards some role's domain of questions. This severely complicates the maintenance of an organization's process descriptions because changes and additions have to be accurately and consistently made to a possibly large number of handbooks.

The notion of Electronic Process Guides (EPGs)[16] was developed to address these problems. The general purpose of an EPG is to guide software developers in doing their tasks by providing fast access to the information they need (e.g., to activity and artifact descriptions, to assets such as checklists, etc.). As implied by its name, an EPG provides online, internet/intranet descriptions of the process. However, rather than merely being electronically provided versions of paper-based handbooks, EPGs are extensively hyper-linked to facilitate navigation through the information about the process. Moreover, EPGs are automatically generated from information captured, non-redundantly, in a process model. Because they are automatically generated, the maintenance of a consistent set of handbooks is much easier and less error prone – any change to the information in the process model is automatically reflected in all the handbooks (once they are regenerated). This makes it much more feasible to provide a set of handbooks with each essentially filtering the information and rendering it in a way most meaningful to some role's questions. Finally, the process model may be formal, rather than informal, and this introduces a rigor that not only precludes common errors (such as using two different names for a process element) but also enables checking the consistency and completeness of both the dynamics and the specification of the process. As a result, EPGs provide up-to-date, accurate, consistent and complete information about a process packaged into the different forms needed to support the different roles in a process. Experience has shown that this considerably enhances the efficiency and effectiveness of process performers.

To explain EPGs further, and indicate their value, we describe two tool suites which have been developed to provide an EPG capability. The first is the SPEARMINT®/EPG tool suite developed at the Fraunhofer Institute for Experimental Software Engineering (IESE)[17]. The second is the Process Management Capability (PMC) tool suite developed at TeraQuest Metrics (TQ)[18]. Both were designed to provide support for managing large process models and the automated generation of online documentation of process handbooks from these process models. They share many common characteristics, but differ in terms of some of the capabilities they provide. In the following, we first describe their common characteristics and then their key differences.

In both cases, the primary objectives were to: (1) support the development of a process model, (2) eliminate common errors, (3) support analysis of the completeness and consistency of both the process and its descriptions, and (4) support the generation of a handbook providing process performance guidance. The tool suites took similar approaches to meeting these primary objectives:

- A well-defined, Entity-Relationship-Attribute style, process modeling technique is used to capture process information. The process model reflects the elementary "information chunks" pertinent to many different question domains (for example, an information chunk that identifies all the roles participating in a task).
- A well-defined data-storage format is used to capture, non-redundantly, the basic "elemental facts" of which information chunks may be composed. For example, the fact that a specific role participates in a specific task would be stored as a single, elemental, fact although it may appear in many different information chunks.
- An editing tool allows a process engineer to capture and maintain the information about a process by establishing and modifying the elemental facts. The editor allows the process engineer to change a set of inter-related facts about the process in a well-coordinated way. The editor may also enforce process modeling rules, for example, the rule: `every task must have at least one participating role`. Finally, the editor allows some degree of customization of a process model through the definition of additional process-element attributes.
- A publishing tool supports the generation of a set of web pages which constitute a process handbook.
- A viewing tool supports the generation of reports (most often also in the form of web pages) supporting a process engineer's work. This

includes, for example, reports concerning inconsistencies and incompleteness.

The differences follow from the fact that the SPEARMINT®/EPG tool suite is focused on the process engineer's need to easily model and analyze a process during its development whereas the PMC tool suite is focused on the process engineer's need to customize an EPG's look-and-feel to the needs and desires of a particular organization. As a result, in the SPEARMINT®/EPG tool suite:

- The editing tool uses a graphical notation for describing processes. It distinguishes the following types of process elements: activities, artifacts, roles, and tools. The notation allows graphically denoting relationships among the process elements as well as the attributes defining the elements' measurable characteristics.
- The viewing tool supports a wide variety of software development and process engineering views. These were defined specifically to support distributed process planning by providing the appropriate representations for reviews.

In the PMC tool suite, however:

- The process model (and editing tool) may be customized to an organization's specific process architecture, i.e., the notions used to describe the organization's processes. The tool suite is based on the Collaborative Process Enactment (COPE)[19] generic process architecture which defines process entity categories (activities, roles, artifacts, conditions and assets) as well as some basic attributes and relationships. It may be customized to reflect an organization's process entity types, their attributes and their relationships.
- The publishing tool is controlled by templates for the various kinds of web pages that appear in a generated handbook. Each template describes the format for the page and identifies the elementary facts that need to be retrieved to generate an instance of the page. A template also describes the computation needed to infer information — for example, information about the flow of artifacts among tasks — from the elementary facts. Multiple sets of templates could be used to gen-

erate multiple handbooks differing either in their look-and-feel or in their role orientation.

Taken together, the two tool suites have been used to create web-based documentation for more than two dozen processes (examples are discussed in several papers[20, 21, 22, 23]; additional examples may be found at *http://www.iese.fhg.de/vincent/examples*). Most of these have been software development processes; a few have concerned an organization's business processes. The experience gained in preparing these EPGs has indicated that both the modeling/analysis enhancements provided by SPEARMINT®/EPG and the EPG-customization enhancements provided by PMC are necessary (and, as a result, the two tool suites are currently being integrated). The experience has also demonstrated the benefits of the EPG approach to process documentation: processes may be developed more rapidly; common errors may be precluded; consistency and completeness may be more extensively and accurately verified; and the time/effort required to maintain and deploy accurate, up-to-date process descriptions is reduced.

Two additional, potential advantages have been noted but not yet fully realized in either tool suite. First, process models are an appropriate means for storing software development knowledge. In general, reusing experience is a key to systematic and disciplined software engineering. Although there are some successful approaches to software product reuse (e.g., class libraries), all kinds of software-related experience, especially process-related experience, should be reused. Process models can be used to capture process-related experience, and this experience can be stored using various structures for an experience repository (e.g., type hierarchies, clusters of domain specific assets). Second, an EPG-based approach to process development allows several kinds of (automated) analyses, which may be performed before the project starts, during process performance and in a post-mortem fashion after project termination. Process models can, for example, be analyzed statically (e.g., to check for consistency) or dynamically (e.g., to check the flow of artifacts across their interface). The latter is important during the modeling of the interfaces of distributed processes.

3.2 Role-based Workspaces

Process handbooks, whether they are created by hand or by using an EPG-based approach, certainly provide the ability for an agent to effectively, efficiently and accurately occupy an assigned role. The major reason is that they contain the information that agents need to answer questions they have before starting to work on, or while working on, their assigned tasks.[c]

Agents also need the ability to focus their attention on the information pertinent to a specific task or a specific question. Locating and organizing the needed information can be extremely difficult no matter how well a role-specific process handbook is designed. In this sub-section, we introduce an approach for focusing on the information pertinent to a specific task. The aim of this is to effectively support project members. When focusing on supporting project managers, a different approach is needed. We discuss this different approach in more detail in section 3.3, when talking about project control.

3.2.1 Role-based Workspace Purpose

We define a *role-based workspace* to be a working environment supporting an agent occupying a role during process performance. A workspace has three major intents. One is to provide access to the documents, tools and assets the agent needs to perform tasks. The second is to help the agent assure that his/her work is satisfactory. The third is to facilitate collaboration with other agents occupying other roles. The first two are discussed in the remainder of this section. The third is discussed in section 4.1.

[c] Up to this point in the chapter, we have talked in terms of processes being composed of *activities*. To discuss role-based workspaces, however, we use the notion of *tasks*. Tasks differ from activities in that: an activity is composed of tasks, a task is an assignable work package, and a task is (almost always) assigned to one role with the role having responsibility of assuring that the task is successfully carried out. We make this distinction because role-based workspaces are intended to help agents complete their assigned tasks in the context of activities carried out by some group of agents (i.e., some team).

A workspace is analogous to a desk in a physical working environment. On or in the desk will be the documents the agents are working on, the tools they use in doing and checking their work, devices for communicating with other agents, and items that help the agents organize their work. More specifically, a workspace corresponds to a desk dedicated to one of the agent's assignments, and an agent would have several desks, one for each of his/her assignments. This indicates a primary benefit of workspaces: the elimination of context-switching overhead. To switch contexts — to switch from working on one assignment to working on another — the agent moves from one desk to another; and when turning attention to an assignment, the agent finds the desk associated with this assignment the same as when he/she last left it. Further, because the desk is assignment-specific rather than agent-specific, much of the overhead associated with delegating or re-assigning work may be eliminated.

To better support today's rapidly growing information-intensive workforce, many of the items on/in a desk are typically moved to a computer. Workspaces can be viewed as an attempt to move as many items as possible (as well as reasonable) off the desk and into the computer. The resulting benefit is again a reduction of overhead effort. Providing electronic access, and allowing automated support, eliminates many assignment-specific overhead activities (for example, searching through a pile or file of documents) and greatly simplifies others (for example, finding the tools needed to work on a document).

3.2.2 Role-based Workspace Organization

A role-based workspace may be either *specific* or *generic*. Both focus on a particular project and a specific role for that project. A specific workspace additionally focuses on a specific task (or small subset of highly coupled tasks) that agents must carry out when occupying the role. Generic workspaces, on the other hand, reflect information pertinent to all the tasks in a project that are pertinent to the role.[d]

---

[d] Workspaces provide access to task-, role- and project-specific information. When working on a task, an agent additionally needs access to relevant personal and corporate-wide information. Relevant personal information includes the agent's schedule and to-be-done list, information that is typically stored in an

An agent accesses a specific workspace by selecting a task (or a strongly coupled subset of tasks). This creates a more narrowly scoped workspace reflecting just this task (or subset of tasks). The agent uses this specific workspace to carry out the work needed to perform the task(s).

An agent opens a generic workspace by first selecting a project to work on and then selecting a role within that project to occupy. The projects that the agent may select are constrained by the allocation of the agent to projects. The roles that may be selected reflect not only resource allocation decisions by the project's manager but also any role-occupancy constraints specified in the role's definition. As an example of the latter, an agent may not open a generic workspace that requires knowledge of a specific analysis tool if the agent's description does not indicate that the agent has this knowledge.

When a workspace is opened, information relevant to the specific agent, the chosen role and the chosen project is assembled and displayed in an organized way. The displayed workspace reflects all the aspects of the process relevant to the role. The contents and organization of the generic workspace reflect information about the agent's abilities, skills and preferences.

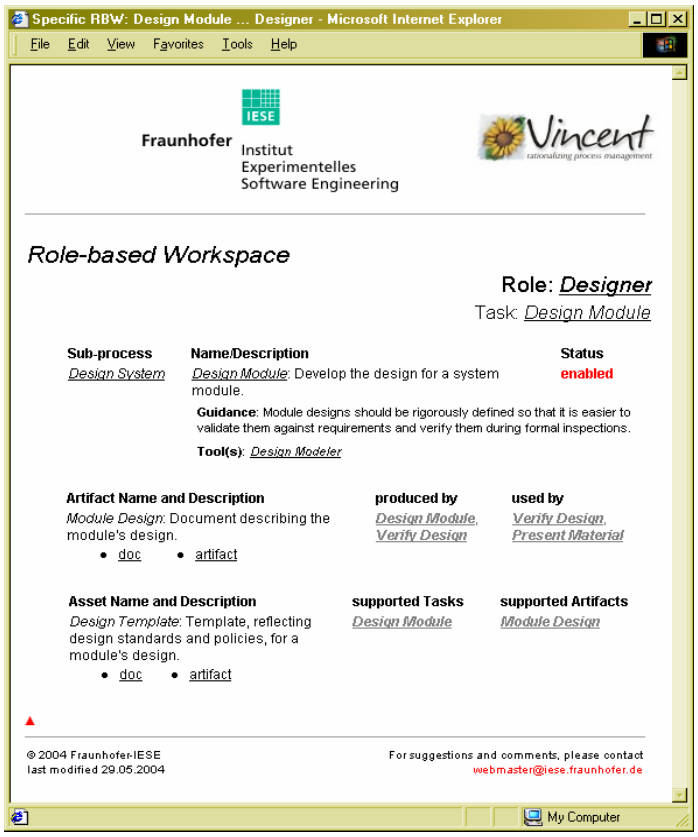
3.2.3 Specific Workspaces

Specific workspaces provide the basic, fundamental support for working on assignments. In the simplest situation, only one agent is working on the task. Frequently, however, two-or-three agents may collaboratively work on the task. In this section, we discuss the simplest, single-agent, situation. The more complex, multi-agent, situation is discussed in section 4.1.

Fig. 3 depicts an example of a specific, single-agent workspace.

---

agent's PDA. Relevant corporate-wide information includes: contact information for other personnel, information about training courses for various tools and techniques, and the identification of personnel who have volunteered to mentor with respect to various tasks or tools. Here, we assume that access to this information is provided outside of a workspace (for example, in an organization's corporate-wide knowledge base).


Specific RBW: Design Module ... Designer - Microsoft Internet Explorer
File Edit View Favorites Tools Help
IESE
Fraunhofer
Institut Experimentelles Software Engineering
Vincent
rationalizing process management
My Computer

Role-based Workspace

Role: Designer

Task: Design Module

Sub-process	Name/Description	Status
Design System	*Design Module*: Develop the design for a system module.	enabled

Guidance: Module designs should be rigorously defined so that it is easier to validate them against requirements and verify them during formal inspections.

Tool(s): *Design Modeler*

Artifact Name and Description	produced by	used by
Module Design: Document describing the module's design. • doc • artifact	*Design Module*, *Verify Design*	*Verify Design*, *Present Material*

Asset Name and Description	supported Tasks	supported Artifacts
Design Template: Template, reflecting design standards and policies, for a module's design. • doc • artifact	*Design Module*	*Module Design*

© 2004 Fraunhofer-IESE
last modified 29.05.2004

For suggestions and comments, please contact
webmaster@iese.fraunhofer.de

Fig. 3. Example of a Specific Role-based Workspace.

This example illustrates the basic items that constitute a specific workspace:

- Task Descriptions: Descriptions of the steps involved in carrying out the task as well as inter-dependencies among the steps. This provides the agent with the basic information about what has to be done when carrying out the task.
- Advice: Guidance concerning how to carry out the task successfully, including lessons learned and the experiences of agents who previously occupied the role. This allows the agent to benefit from previous experience in carrying out the task.

- Artifacts: Descriptions of the artifacts that the role works on when carrying out the task as well as links to the actual artifacts needed when or resulting from performing the task. This information allows the agent to understand the task, and review the result of performing it, in terms of its effect on the artifacts used and produced during task performance.
- Tools: Descriptions of the tools the agent needs in performing the task as well as links to the tools' documentation and the tools themselves. This information allows the agent to efficiently access the tools and consult tool documentation. It can also guide the agent in choosing among a collection of alternative tools.
- Assets: Descriptions of relevant templates, checklists, etc., as well as links by which to download copies of these assets or view policy, guideline, standards and reference documents. This information provides the agent with a task-specific "doorway" into the usually very large collection of assets the organization has accumulated over time.

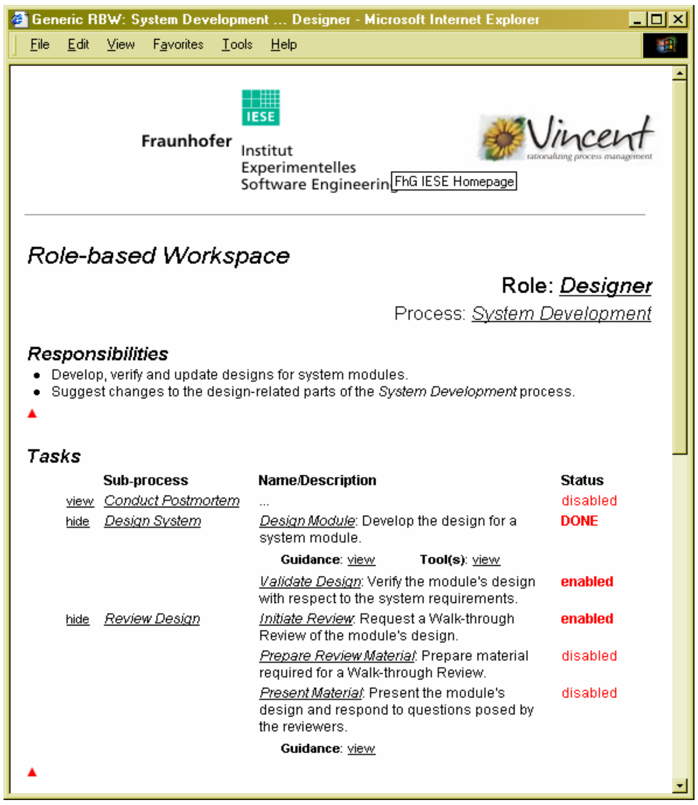
3.2.4 Generic Workspaces

A specific workspace provides the basic support an agent needs to rationally carry out a specific task. A generic workspace, on the other hand, is role-specific but does not pertain to any specific task. It reflects information about the role in general and the full complement of tasks in which the role participates. Its major intent is to help agents occupying the role to properly achieve the role's purpose in a process and properly function as a member of the project team performing the process.

Fig. 4 and Fig. 5 illustrate a generic workspace. These figures show that, to assist agents, a generic workspace includes information about:

- Responsibilities: Descriptions of the role's overall obligations and permissions with, where possible, links to the tasks, artifacts, and other process elements pertinent to meeting the obligations within the constraints levied by the permissions. This helps the agent understand and focus on his/her responsibilities.

- Task List: A list of the tasks for which the role is responsible. As new tasks are assigned they are added to this list. The list provides the role with a continuously up-to-date agenda for his/her work.
- Process Documentation: A structured collection of links into relevant parts of the process handbook. This allows the agent to efficiently consult the process description as needed in the course of his/her work.


Generic RBW: System Development ... Designer - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Fraunhofer
IESE
Institut Experimentelles Software Engineering
FhG IESE Homepage
Vincent
rationalizing process management

Role-based Workspace

Role: *Designer*

Process: *System Development*

Responsibilities

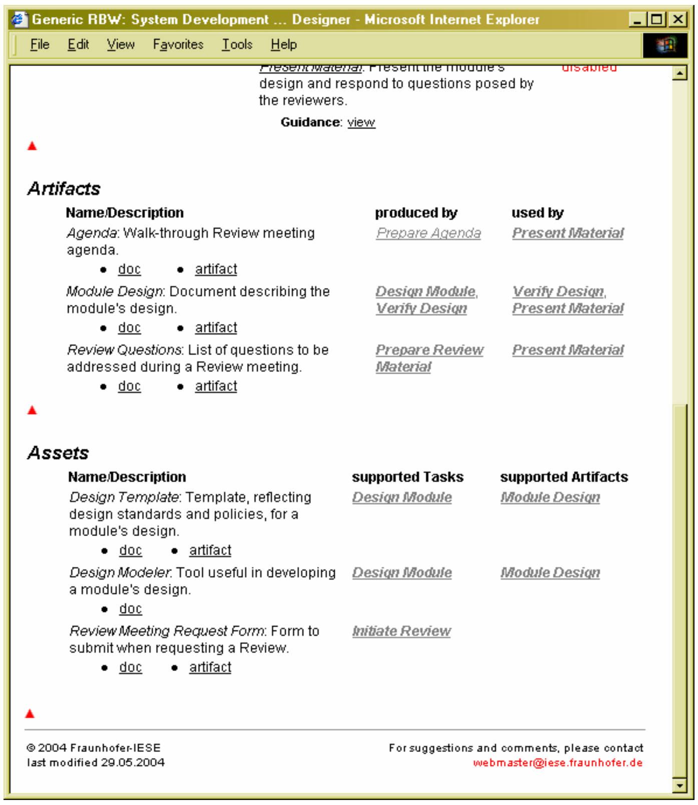
- Develop, verify and update designs for system modules.
- Suggest changes to the design-related parts of the *System Development* process.

Tasks

	Sub-process	Name/Description	Status
view	*Conduct Postmortem*	...	disabled
hide	*Design System*	*Design Module*: Develop the design for a system module. Guidance: view Tool(s): view	DONE
		Validate Design: Verify the module's design with respect to the system requirements.	enabled
hide	*Review Design*	*Initiate Review*: Request a Walk-through Review of the module's design.	enabled
		Prepare Review Material: Prepare material required for a Walk-through Review.	disabled
		Present Material: Present the module's design and respond to questions posed by the reviewers. Guidance: view	disabled

Fig. 4. Example of a Generic Role-based Workspace (Responsibilities and a selection of assigned Tasks).

This information not only helps agents plan and track their work, but also helps them better understand the constraints upon their work and the rationale for this work as part of the overall process being performed.


Generic RBW: System Development ... Designer - Microsoft Internet Explorer
File Edit View Favorites Tools Help

Present Material: Present the module's design and respond to questions posed by the reviewers. disabled

Guidance: view

Artifacts

Name/Description	produced by	used by
Agenda: Walk-through Review meeting agenda. • doc • artifact	Prepare Agenda	Present Material
Module Design: Document describing the module's design. • doc • artifact	Design Module, Verify Design	Verify Design, Present Material
Review Questions: List of questions to be addressed during a Review meeting. • doc • artifact	Prepare Review Material	Present Material

Assets

Name/Description	supported Tasks	supported Artifacts
Design Template: Template, reflecting design standards and policies, for a module's design. • doc • artifact	Design Module	Module Design
Design Modeler: Tool useful in developing a module's design. • doc	Design Module	Module Design
Review Meeting Request Form: Form to submit when requesting a Review. • doc • artifact	Initiate Review	

© 2004 Fraunhofer-IESE
last modified 29.05.2004

For suggestions and comments, please contact
webmaster@iese.fraunhofer.de

Fig. 5. Example of a Generic Role-based Workspace (Artifacts and Assets related to viewed Tasks).

3.3 *Software Project Control Centers*

The complexity of software development projects is continuously increasing. This results from the ever-increasing complexity of functional as well as non-functional software requirements (e.g., reliability or time constraints for safety critical systems). The more complex the requirements are, the more people are usually involved in meeting them, which only further increases the complexity of controlling and coordinating the project. This, in turn, makes it even harder to develop the system according to the plan (i.e., matching time and budget constraints). Coordination issues are usually addressed by Computer Supported Cooperative Work

(CSCW) systems which ease communication among project members and support document sharing. CSCW support is further discussed in section 4.1. The focus in this section is on support for controlling a project.

Project control issues are very hard to handle. Many software development organizations still lack support for obtaining intellectual control over their software development projects and for determining the performance of their development processes and the quality of the produced products. Systematic support for detecting and reacting to critical project states in order to achieve planned goals is usually missing[24].

One way to support effective control of software development projects is the use of basic engineering principles[25, 26], with particular attention to the monitoring and analysis of actual product and process states, the comparison of actual with planned states, and the initiation of any necessary corrective actions during project execution. Effectively applying these principles requires experience-based project planning[27]; that is, the capture of experience from previous projects (such as activities, measurement plans, and baselines), and the use of explicitly defined models reflecting this experience, in order to plan a project. Furthermore, it requires the collection, interpretation, and presentation of measurement data according to a measurement plan; that is, the establishment of measurement-based feedback mechanisms in order to provide stakeholders with up-to-date information about the project state. Moreover, it requires experience packaging after project completion so that future projects are influenced by the experience gained in previously-performed projects.

In the aeronautical domain, *air traffic control systems* are used to ensure the safe operation of commercial and private aircraft. Air traffic controllers use these systems to coordinate the safe and efficient movement of air traffic (e.g., to make certain that planes stay a safe distance apart or to minimize delays). The system collects and visualizes all critical data (e.g., the distance between two planes, the planned arrival and departure times) in order to support decisions by air traffic controllers. Software project control requires an analogous approach that is tailored to the specifics of the process being used (for example, its non-deterministic, concurrent, and distributed nature).

A *Software Project Control Center* (SPCC)[24] is a control system for software development which collects all relevant data to project control, interprets and analyzes the data according to the project's control needs, visualizes the data for different project roles, and suggests corrective actions in the case of plan deviations. An SPCC could also support packaging of data (e.g., as predictive models) for future use and contribute to an improvement cycle spanning a series of projects.

Before discussing existing SPCC approaches, we first discuss the notion of "controlling a project". *Controlling a project* means ensuring the satisfaction of project objectives by monitoring and measuring progress regularly in order to identify variances from plan during project execution so that corrective action can be taken when necessary[28]. Planning is the basis for project control and defines expectations which can be checked during project execution. The gathered experience can be packaged for future projects after project completion in order to support organization-wide improvement cycles. All corrective actions needed to bring a project back to plan – that is, all *steering activities* – are explicitly included in the notion of "controlling a project".

A Software Project Control Center is a means for interpretation and visualization of measurement data during process performance and therefore supports controlling a project. An SPCC has a logical architecture that clearly defines interfaces to its environment, especially to all project members relying on SPCC information, and a set of underlying techniques and methods that support controlling a project.

From a more technical perspective, an SPCC utilizes data from the current project (e.g., the project's goals, characteristics, baselines, and measurement data) and experiences from previous projects (e.g., information captured in quality, product, and process models) and produces a visualization of measurement data by using the incorporated techniques and methods to interpret the data. An SPCC is a general approach to project control and is not necessarily tool-supported. But in order to successfully, efficiently carry out control activities such as monitoring defect profiles, detecting abnormal effort deviations, cost estimation, and root-cause analyses of plan deviations, a certain amount of tool support is necessary and inevitable.

In the following we highlight two SPCC approaches addressing different objectives. The first deals with integrated approaches; that is, SPCC approaches that are tightly integrated into the project's performance and act as a focal point for all project issues and organizational improvement efforts. The second deals with goal-oriented data processing and visualization; that is, presenting data regarding different project needs and supporting different project stakeholders.

3.3.1 Integrated Controlling Approaches

Integrated SPCC approaches are tightly integrated into every project's performance and are actively used to gain experience for future projects. Such approaches are normally used by organizations to improve the maturity of their software processes and practices and establish organization-wide standards.

One example is NASA's *Software Management Environment (SME)*[29, 30], which was developed by the Software Engineering Laboratory (SEL)[31, 32] at the NASA Goddard Space Flight Center (GSFC). The main aim of this SPCC is to support the manager of a software development project by providing access to three information sources: (1) The *SEL Database* holding information from previous projects; that is, subjective and objective process and product data, plans, and tool usages. (2) The *SEL Research Results* database holding different models (such as growth or effort models) and relationships between certain parameters/attributes (described with quality models). Primarily, this information may be used to predict and assess attributes. (3) The *SEL Management Experience* database holding information about project management experiences in the form of rules within an expert system. The rules help inexperienced managers analyze data and guide re-planning activities. For example, this database includes lists of errors and appropriate corrective actions.

All this information is input for an SME, which uses it to perform management-oriented analyses fostering well-founded decision-making. Experience gained during project execution may lead to changes of the information. This feedback mechanism enables the SME to work with up-to-date information.

3.3.2 Goal-oriented Data Visualization

A Goal-oriented SPCC approach (G-SPCC) is a state-of-the-art framework for project control developed at the University of Kaiserslautern and the Fraunhofer Institute for Experimental Software Engineering (IESE)[33, 34]. The aim of this approach is to present the collected data in a goal-oriented way in order to optimize a measurement program and effectively detect plan deviations.

The purpose of the G-SPCC approach is to support agents occupying roles. Project control is driven by different role-oriented needs. We define *control needs* as a set of role-dependent requirements for obtaining project control. A project manager needs different kinds of data, data of different granularity, or different data presentation modes than a quality assurance manager or a developer. For example, a manager is interested in an overview of the project effort in order to compare it to previously defined baselines, while a developer is interested in the effort she/he spent on a certain activity. As another example, a quality assurance manager is interested in the efficiency of a certain inspection technique, while the project manager is primarily interested in a list of defects and how many defects have to be fixed in order to release a product to the next project phase. In general, control-oriented information needs differ between more management-oriented project roles (such as a project manager or a quality assurance manager) and more technically oriented roles (such as a tester or a programmer). The first group is more interested in charts presenting an overview of the overall project, while the second is more interested in activities within the project. However, it is important to note that control-oriented information needs may vary, significantly, within these groups.

Fig. 6 gives an overview of the G-SPCC architecture. It shows that measurement data is collected during project performance and interpreted with respect to the goals and characteristics of the project as well as project plan information (e.g., baselines, number of project members, and developer skills) and control needs (e.g., the kind of control technique that should be applied, and tolerance ranges). The outputs of this interpretation (performed by *SPCC functions*) are displayed by a set of *SPCC views*, each providing role-specific insights into the process (e.g., insights suitable for project managers, quality assurance personnel, or the development group). The SPCC interpretation and visualization process is supported by an experience base in order to reflect data from previous projects and store experience gathered after project completion.

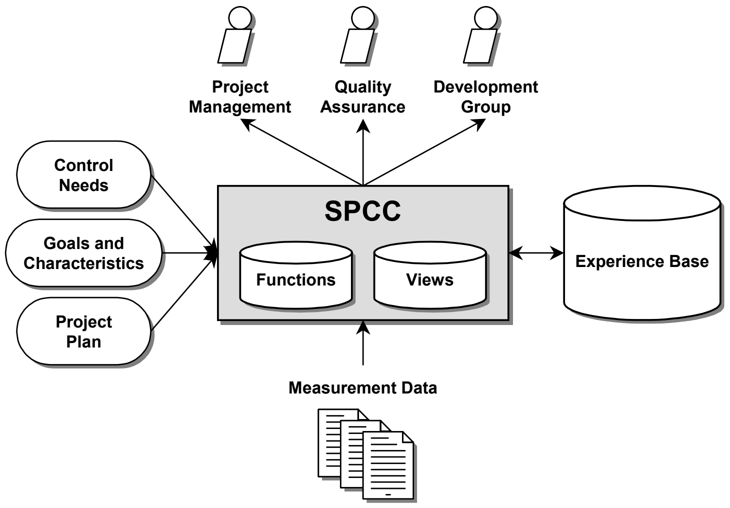


Fig. 6. The G-SPCC Architecture to Support Different Project Roles.

The G-SPCC approach is based on the Quality Improvement Paradigm (QIP)[35] and consists of the following steps:

- First, project stakeholder control needs are characterized in order to set up a measurement program able to provide a basis for satisfying all needs.
- Then, measurement goals are defined and metrics are derived determining what kind of data to collect. The GQM paradigm is used to derive these metrics and create a set of data collection sheets that are assigned to certain process steps. The process is modeled using SPEARMINT®/EPG (described in section 3.1).
- Next, a *Visualization Catena* (VC) is defined to provide online feedback on the basis of the collected data. The VC includes a set of control techniques and methods corresponding to the measurement goals. For example, a suitable Tolerance Range Checking technique may be

included to detect baseline deviations. A VC also includes a set of views to visualize project data.

- Once the VC is specified, a set of role-oriented views are defined to support control of the project. As measurement data are collected, the VC analyzes and visualizes them accordingly. For example, an SPCC function detects baseline deviations and a corresponding view displays them.
- Once a deviation is detected, its root cause must be determined and the VC adapted accordingly. A baseline deviation, for example, can lead to new or adapted measurement goals and baselines. In this case, new VC components are defined for an existing VC or existing components are changed or removed.
- After project completion, the resulting VC may be used to improve future VCs, software development processes, and baselines. For example, views of the generalized effort progression can be used to improve the baselining of future projects or process efficiency views may be used to enhance the definition of process steps.

The benefits of the G-SPCC approach include: (1) improvement of quality assurance and project control by providing a set of custom-made views of measurement data, (2) support of project management through early detection of plan deviations and proactive intervention, (3) support of distributed software development by establishing a single point of control, (4) enhanced understanding of software processes, and improvement of these processes, via measurement-based feedback, and (5) preventing information overload through custom-made views with different levels of abstraction.

4 Using Process Information

The purpose of this section is to illustrate how displayed process information may be used to support process performers in doing their work. First, the usage of role-based workspaces (introduced in Section 3.2) is discussed, including their customization and personalization as well as coordination and collaboration issues. Second, some sample control techniques based on a Software Project Control Center (introduced in section 3.3) are discussed. Subsequent to this, two sections briefly address more advanced concepts, namely the improvement-oriented organization of experience (including process information) and support for process enactment (that is, providing more proactive support for enacting a defined process).

4.1 Using Role-based Workspaces

Previously, we discussed role-based workspaces as a way to collect and organize the information pertinent to an agent when occupying a role and performing some task. In this section, we first discuss the tailoring of role-based workspaces to meet the needs and capabilities of specific role occupants. We then discuss the various ways in which role-based workspaces may provide significant support for coordination and collaboration, helping an agent perform more effectively, efficiently and accurately as a member of a project team.

4.1.1 Customization

Different agents will have different capabilities stemming from their differing levels of education, training and experience. Effective role-based support requires that role-based workspaces may be customized to match a role's specific capabilities. Customization involves:

- Varying the content: Inexperienced, novice agents need more support than experienced, expert ones. For example, guidance might be included in a novice's role-based workspace but not included in a role-based workspace for an expert. Also included in a novice's, but not an expert's, workspace might be: detailed rather than synoptic descriptions of the tasks; explanations of required skills; an indication of the factors affecting successful task completion; as well as other information.
- Providing access to available support: Novices will also benefit from being able to easily access support for their work. This could include an indication of professional development courses related to a role's

required skills. It could also include an identification of role-related mentors accompanied by their contact information.

- Varying the display of information: Agents will vary with respect to their basic approach to solving problems. An often-noted, and very major, difference is that between graphically-oriented problem-solvers versus textually-oriented ones. This implies that different modes should be available for displaying process-related information. For example, an activity decomposition structure might be displayed as hierarchically indented text or graphically as a tree.
- Varying the organization of information: Another often noted, and also major, difference among agents is whether they approach problem-solving in a bottom-up (inductive) or top-down (deductive) manner. This also implies that different information-display modes should be available. For example, drop-down lists might be available to allow the agent to navigate through the information in a top-down (deductive) manner.

Obviously, information about an agent's capabilities is needed in order to provide customized role-based workspaces. Much of this information can come from providing agent profiles identifying expected collections of capabilities and having the agent identify the most appropriate profile when establishing a role-based workspace. Alternatively, the program which establishes a role-based workspace could hold a pre-programmed dialog with the agent to obtain information affecting the workspace's content, organization, and rendition.

4.1.2 Personalization

Agent capability differences will also lead to different patterns of role-based workspace usage. In addition, agents will accumulate – over time – their personal arsenal of resources they have found useful in carrying out their work. Finally, agents will gather observations and advice about how to effectively, efficiently, and accurately carry out the work.

Therefore, effective role-based support requires that agents be able to personalize their role-based workspaces. Personalization involves:

- Re-defining the content: It is impossible to predict all the ways in which role-based workspaces should be customized to meet individual agent needs. It should be possible for agents to delete information from and add information to the workspace.
- Re-organizing the display: It is also impossible to predict all the ways individual agents will want to position the information displayed in a workspace. It should be possible for agents to combine, split and reposition the information within the role-based workspace's display.
- Adding assets: Agents will, over time, accumulate many personal assets they have found valuable in performing their work. It should be possible for agents to add these assets to their workspaces.
- Annotating the displayed information: In analogy to writing marginal notes in a paper-based process handbook, it should be possible for agents to add notes to pieces of information in their workspaces.

Supporting personalization along these lines requires a variety of workspace support capabilities. Many of these capabilities imply the inclusion of a variety of ways to display information on desktops and within windows; providing these capabilities depends on the capabilities provided by the support system's underlying operating-system contexts (e.g., MS 2000 vs. MS 2003). Others concern the manipulation of information provided by web page displays. Regardless of how they may be achieved, the capabilities include:

- Tracking of desktop and window contents and organization accompanied by reconstitution of the configuration when the desktop or window is re-opened.
- Drag-and-drop manipulation of information displayed on a desktop and within windows (for example, repositioning text or other elements within the display).
- Drag-and-drop manipulation of information displayed within web page displays (for example, repositioning the items in some graphic included in a web page).
- Annotations of documents and web-pages (for example, providing text input boxes within a web page).

- Dynamic changes to information display modes (for example, changing from a drop-down list to a tiled set of windows).

All of the cited example capabilities are provided by current, state-of-the-practice technology. In fact, current technology might support even more extensive capabilities and even more effective workspace personalization. Empirical studies of the possibilities, and their importance, is needed to determine what can be provided and its value.

4.1.3 Coordination

A major benefit of role-based workspaces is that they can support the coordination of work either by an agent over time or among a group of agents carrying out inter-related tasks. By coordination we mean the effective, efficient, accurate performance of tasks in an orderly fashion as affected by inter-task constraints. We discuss this benefit in this section.

A very simple case of coordination concerns the ability of an agent to easily switch among the various roles he/she may occupy. An agent may have several generic and specific workspaces, perhaps from different projects, open at some point in time. However, only one specific workspace and its relevant generic workspace will be *active*. This helps agents work on assignments in a well-focused way. It also helps them rapidly and efficiently switch among their various assignments. Current operating-system contexts fully support this *context switching*.

More significant is the support role-based workspaces can provide for organizing and guiding an agent's work within the context of a process. A particularly useful, yet quite simple, situation is helping the agent focus only on those tasks that he/she <u>may</u> work on. As indicated previously, the generic workspace includes a list of the tasks the role is responsible for or participates in. This list may indicate which tasks are *enabled* — may be worked on — and which tasks are *blocked* — may not be worked on because some pre-condition is not satisfied. This allows the agent to quickly focus his/her attention. (Fig. 3 provides an example of displaying information about the *enabled* status of tasks.)

Additional support along these lines may focus on other aspects of carrying out the tasks. This support relates to the status of the tasks and can include:

- Condition Checkers: These are tools the agent may use to check the completeness, correctness and accuracy of his/her work. Process descriptions will often identify, usually in the terms of the status of artifacts, conditions that must be achieved, for example, that a `Design` activity must result in a `well-organized Design` artifact. To the extent that the `Design` being `well-organized` can be checked by analyzing the `Design`, the agent can receive assistance in checking this condition.
- Events: Often, conditions may not be checked in an automated way. For example, the condition that a `Design` document is `correct` is usually checked by a combination of desk-checking and design reviews. To support task enabling and blocking, the fact that the `Design` document is `correct` needs to be recorded. A role-based workspace should provide the ability to record the satisfaction of conditions so that this information may be used to coordinate the agent's future work.
- Task Dependencies: It is often the case that the tasks being performed by one agent are enabled by the work completed by other agents – this is, in fact, the most usual situation. Therefore events, in general, lead to the enabling of tasks in a workspace. This leads to changes in the enabled/blocked status of the tasks in some other workspace. Active "announcement" of status changes – for example, by some visual or aural signal such as those typically used to announce "you have mail" – will help the agent keep up with changes to his/her work.

In summary, one major purpose of coordination capabilities is to allow agents to assess the completeness and accuracy of their work.

Another major purpose is to coordinate work across several tasks being carried out by several agents. This requires capabilities that allow agents to signal each other about the status of their work. This could lead to quite extensive coordination support. For example, if a task is blocked, the agent might use the Process Documentation items in a generic workspace to track back through pre-condition/post-conditions to locate predecessor tasks and then use a Task Status Query to get information about the source of the blockage. If necessary, the agent may then interact with other agents (using capabilities discussed in the following section) to collaboratively understand and solve problems inhibiting progress. This allows the agents to collectively work in a very focused way, making their collective work more effective and efficient.

There is an additional possibility for supporting cross-agent coordination. In the task list, additional information may be displayed about enabled tasks to indicate tasks for which the satisfaction of one or more pre-conditions has been re-established since the agent previously completed the task. This flags tasks the agent might have to re-do because some process element, probably an artifact, has been changed. The agent may open specific workspaces for these flagged tasks, use task dependency information in the workspaces or process handbooks to understand what has changed, and carry out the task again as needed.

4.1.4 Collaboration

Inter-agent coordination support is particularly valuable when the agents are geographically distributed. Event-based enabling of items on task lists provides significant coordination support in this case. But much more is needed – the agents additionally need support for the collaboration that, were they geographically co-located, would be accomplished by organized or informal face-to-face meetings and discussions of their work and any problems which arise.

Two possibilities arise when considering collaboration within a group of geographically separated agents. One case – *synchronous collaboration* – occurs when the agents can all work at the same time. In the other case – *asynchronous collaboration* – the agents must for some reason (availability, time zone differences, etc.) work at different times. We discuss the use of role-based workspaces to support synchronous and asynchronous collaboration in the remainder of this section.

One of the major goals of work in the field of Computer Supported Cooperative Work (CSCW) has been to support geographically distributed teams working synchronously. (For a general introduction to CSCW work, see the overview by Grundin et al.[36].) CSCW work suggests that this support includes at least the following basic capabilities:

- Agenda: A meeting agenda provides both a plan for a specific real-time meeting and a place to record decisions and action items.

- Audio: Support for communication by voice.
- Shared Windows: Display of information at multiple various workstations with changes made by the "owner" of the displayed information propagated to the displays.
- Shared Whiteboard: A window displayed on all the workstations that all participants can modify using drawing/text capabilities they would typically use in writing on a whiteboard in a meeting room.
- Real-time Chat: Broadcasted and directed, person-to-person, transmission of commentary to allow participants to record their thoughts and share them with others.

Additional capabilities may be included to support specific needs. For example, a video image capture/display capability might be added to allow broadcasting of a (physical) whiteboard in one of the agents' offices.

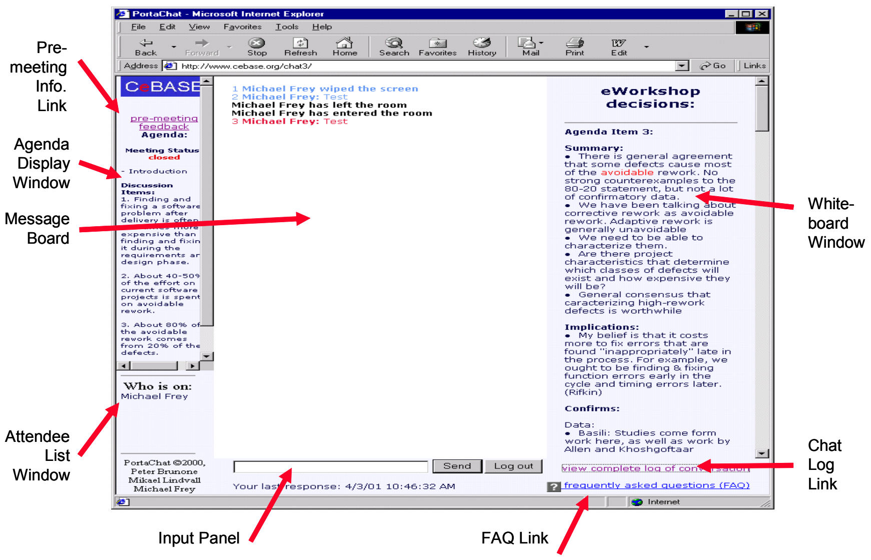
These basic capabilities could be added to a specific workspace to support synchronous collaboration. CSCW work to date, however, provides a much simpler solution. This work has led to many distributed meeting systems, several of which are commercially available. Because these focused and integrative solutions exist, generic workspaces do not have to be extended to provide the capabilities. Rather, they may be used in tandem with these other solutions.

Representative commercial products are Microsoft's *NetMeeting*[37] and Teamware's *Pl@za*[38]. Another example is the eWorkshop system developed at the Fraunhofer Institute at the University of Maryland[39]. These distributed meeting systems may be used to provide the needed agenda, audio, shared whiteboard and real-time chat support. An example collaboration support window, resulting from using the eWorkshop system, is shown in Fig. 7.

Collaboration across tasks additionally requires substantial support for coordinating the mutual influences and constraints among the tasks as specified in the process definition. For example, the process description might indicate the flow of artifacts among tasks by indicating how the tasks produce and consume the documents. As another example, the process definition might specify, or at least imply, precedence relations among the tasks.

Collaboration across tasks may be supported by the following basic capabilities:

- Asynchronous Communication: Mail-style interactions among the agents.
- Shared Document Spaces: A shared file structure that all of the agents can access and modify.
- Threaded Discussions: A means to raise questions about, add support to, and refute comments about some issue as well as spawn new issues.


PortaChat - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit
Address http://www.cebase.org/chat3/
Go Links
CeBASE
Send Log out
Internet
Pre-meeting Info. Link
Agenda Display Window
Message Board
Attendee List Window
Input Panel
FAQ Link
White-board Window
Chat Log Link

pre-meeting feedback
Agenda:

Meeting Status
closed

- Introduction

Discussion Items:
1. Finding and fixing a software problem after delivery is often ... more expensive than finding and fixin it during the requirements ar design phase.

2. About 40-50% of the effort on current software projects is spent on avoidable rework.

3. About 80% of the avoidable rework comes from 20% of the defects.

Who is on:
Michael Frey

PortaChat ©2000,
Peter Brunone
Mikael Lindvall
Michael Frey

1 Michael Frey wiped the screen
2 Michael Frey: Test
Michael Frey has left the room
Michael Frey has entered the room
3 Michael Frey: Test

Your last response: 4/3/01 10:46:32 AM

eWorkshop decisions:

Agenda Item 3:

Summary:
- There is general agreement that some defects cause most of the avoidable rework. No strong counterexamples to the 80-20 statement, but not a lot of confirmatory data.
- We have been talking about corrective rework as avoidable rework. Adaptive rework is generally unavoidable
- We need to be able to characterize them.
- Are there project characteristics that determine which classes of defects will exist and how expensive they will be?
- General consensus that caracterizing high-rework defects is worthwhile

Implications:
- My belief is that it costs more to fix errors that are found "inappropriately" late in the process. For example, we ought to be finding & fixing function errors early in the cycle and timing errors later. (Rifkin)

Confirms:

Data:
- Basili: Studies come form work here, as well as work by Allen and Khoshgoftaar

view complete log of conversation
frequently asked questions (FAQ)

Fig. 7. Example of CSCW Support for Role-based Workspaces.

E-mail is the normal approach to supporting asynchronous communication. An example system developed to provide shared document spaces is the Basic Support for Collaborative Work (BSCW) system[40]. Threaded discussions about documents are also supported by BCSW. Support for threaded discussions in general has affected the support provided by most currently available commercial systems. CSCW work has led to systems — Teamware's *Pl@za* is, again, an example — that integrate support for asynchronous communication, shared document spaces, and threaded discussions into systems that support distributed meetings.

4.2 Using Software Project Control Centers

Section 3.3 introduced Software Project Control Centers (SPCCs) as a primary means to support project control and discussed how to provide a role-oriented visualization for gathered measurement data. The focus of this section is on how an SPCC may be utilized by a specific project role; that is, what kind of concrete support can be provided. Before we give examples of this support, we first address the basic purpose of project control; i.e., we address the purposes of an SPCC application.

The following discussion uses concepts adapted from the Software Management Environment (SME) approach[29]:

- *Monitoring*[e] refers to observing a project's state and progress by observing attributes, or combinations of attributes, of the project's processes, products, and resources.
- *Comparison* uses archived data from completed projects or nominal performance guidelines to judge the progress and health of the current project.
- *Analysis* focuses on (1) examining the monitoring results, and (2) applying information about a project's context to identify the probable causes of deviations from the nominal performance guidelines.
- *Assessment* analyzes, with weighting, information about the project to form a judgment of project, product, and process quality.
- *Prediction* extrapolates information about attributes of a project's processes, products, and resources from the current project status to assess the future project behavior. In general, prediction always requires some kind of mathematical model. Fenton and Pfleeger[41] define prediction as identifying relationships between various process and product factors and using these relationships to predict relevant external attributes of products and processes.

[e] In the SME approach this is called *observation*.

	Module A	Module B	Module C	Module D	Total
Planned Value (PV)	20	40	30	10	100
Earned Value (EV)	20	30	10	0	60
Actual Cost (AC)	18	32	17	0	57
Schedule Variance = EV - PV	0	-10	-20	-10	-40
Cost Variance = EV - AC	2	-2	-7	0	-7

Table 1. Earned Value Sample for Module Costs in Thousand US Dollars.

- *Planning* defines baselines or a nominal value for certain measures. In addition, it focuses on assessing (alternative) planning decisions and their effects. This is the basis for further dynamic re-planning during the execution of the project.
- *Guidance* proposes a number of courses of action according to a specific situation or an identified problem. Based on these proposals, a manager might be able to initiate corrective actions and take steps to improve or enhance the development process. A developer, on the other hand, might use guidance as assistance for harmonizing his/her own performance with respect to the overall process and given project goals.

The core of an SPCC is the set of integrated project-control techniques and methods. These methods usually cover different purposes, such as monitoring project attributes, comparing attributes to baselines, or predicting the course of an attribute. The most advanced support includes assessing the overall state of the project and guiding a project participant through corrective actions if a project differs from its plan. Assessing a project's overall state may be achieved by *Earned Value Approaches*[42], which identify important key factors for assessing the overall project state. Project participant guidance may be achieved by building upon experience from previous projects.

4.2.1 Assigning Overall Project State

The Earned-Value Approach is a management technique used to assess the current state of a project. It was first defined at the end of 19th century when engineers decided to determine cost efficiency by comparing the actual cost (AC) of work performed with the earned value (EV) and the planned value (PV).

Table 1 illustrates the basics of this approach: It concerns the control of the costs (measured in thousand US dollars) of creating four software modules, A to D. For each module, we have planned values derived during the planning phase and the actual costs determined by data collection procedures. As work is performed, the earned value of this work is measured on the same basis as for the planned values. Several techniques may be used to compute the earned value for a certain work package; these are beyond the scope of this chapter. However, to provide a simple example: if a work package is complete, its earned value is equal to the planned value, whereas, if a work package has not started yet, its earned value is zero. For example, in Table 1, module A is complete, work on module D has not yet started, and one-third of module C's planned value has already been achieved.

When controlling a project, project managers are interested in the *schedule variance* (that is, the earned value minus the planned value) and the *cost variance* (that is, the earned value minus the actual cost). In the example shown in Table 1, and with respect to the point in the project reflected in the table, work for 40,000 US dollars remains to be done, but 7,000 US dollars have been expended without any recognition of its value. A simplistic conclusion is that the project plan has been violated. An earned-value conclusion, however, would compare the actual cost with the planned values and not indicate a plan violation.

4.2.2 Experience-Based Approach

The first technique we address in this context is the *Sprint I* approach[43, 44, 45], built upon clustering algorithms to dynamically adapt the prediction of key project attributes during project execution. Sprint I is not a pure approach to project control according to our definition because it predicts project attributes before the project starts and, therefore, covers planning as well as performance concerns. However, the Sprint I approach provides an example of how a project manager might use experience from previous projects in order to control an on-going project. The prerequisite for a successful application of Sprint I is that a software development organization has performed a number of similar projects and measured at least one key attribute (e.g., effort per development phase) for each of these projects. Additionally, there is the requirement that the context for each of these projects (i.e., the boundary conditions such as organizational, personal and technical constraints) has been characterized.

Briefly, the technique is as follows: First, context-specific measurement data from former projects is analyzed in order to identify clusters. Based on the context of the project to be controlled, the technique selects a suitable cluster and uses the cluster curve (the mean of all curves within a cluster) for predicting the attributes to be controlled. During the performance of the project, the prediction is modified based on actual project data. This leads to an empirically based prediction and, as a result, to flexibility for project and context changes.

The second experience-based approach to applying SPCCs was developed in the context of NASA's SME. It capitalizes on experience gained in previously-performed projects. Doerflinger and Basili[46] describe the use of dynamic variables as a means to monitor software development and provide guidance in case of plan deviations. The core notion is to assume underlying relationships that are invariant across similar projects. These relationships are used to predict the behavior of projects with similar characteristics. A *baseline* for a certain project attribute (such as the effort in person-hours) is derived from measurement data for one or more completed projects within the same context. The baseline is used to determine whether the project is in trouble or not. If the current values of a variable fall outside a *tolerance range* (i.e., a predetermined tolerable variation from the baseline), the project manager is alerted and has to determine the possible reasons for the failure. A number of tables list possible reasons for deviations above or below the tolerance range for each measured project attribute. If a particular reason appears more often than some other reason, the former is assumed to be more probable than the latter.

4.3 Process Enactment Support

So far, we have focused on enabling, facilitating the work of agents in carrying out their assigned tasks. The capabilities we have discussed serve to present information about what should happen, what has happened and the status of a project in qualitative (status related) and quantitative (measurement based) terms. Agents interpret this information to focus on the tasks they need to do, understand how they can carry out these tasks, and gain access to supportive resources.

More proactive support is possible. With this proactive support, there is some control over focusing the agents' attention, directing his/her work, and promoting the use of specific resources. In general terms, with proactive support: (1) an automated system makes decisions rather than allowing the agents to make decisions, and (2) the automated system implements some of the actions that the agent might carry out.

A simple form of proactive support is *workflow management*. In this case, the flow of artifacts, as defined in the process handbook, is used to actively focus the attention of agents on enabled tasks and actively manage the flow of the actual artifacts as agents complete their tasks. Another, more extensive, form of proactive support is *enactment support*. In this case, the status of, and measurements about, all process elements, not just artifacts, is used to focus agent attention and manage the process performance.

Numerous workflow management and enactment support systems have been developed and they have focused on many software development process issues. The majority of them have been designed to support the performance of a project by tracking states and actively guiding developers based on this state information. Examples range from flexible workflow management systems, to object-oriented database systems, to distributed, open environments. The systems developed to date have been decidedly immature because of the complexity of the goal. Nevertheless, some success stories do exist, such as LEU[47] from LION GmbH, Germany, and ProcessWEAVER[48] from Cap Gemini, France.

4.4 Experience Management

This section discusses the ways in which experience from former projects may be reused for planning and controlling new projects. Basically, we can distinguish among organization-wide experience (such as general effort baselines for a certain project type, lessons learned from former projects, process models, product models, etc.) and project-specific experience (such as project plan information, instances of models, schedules, specific effort distributions, etc.). We define an *experience base* to be a repository of both organization-wide and project-specific experience described with respect to context-specificity and significance, for example, the experience pertains to a specific context (e.g., valid for all projects of a certain domain) and is stated with respect to a certain significance (e.g., a model has been successfully applied in five projects).

One approach to providing an experience base is an Experience Factory[35, 49]. This approach is based on the Quality Improvement Paradigm (QIP) approach to evolutionary process improvement. The QIP approach comprises the following basic steps: (1) Characterize, (2) Set Goals, (3) Choose Process, (4) Execute, (5) Analyze, and (6) Package. Each of the six steps can be interpreted in both a project-specific and organization-wide manner:

- The steps can be characterized as follows for a specific software development project. (1) Characterize the project environment; that is, determine the project type and elements to be reused from an experience base. (2) Set quantifiable goals; that is, define quality goals and select corresponding models, specify hypotheses, and identify influencing factors for the hypotheses. (3) Choose the right process and define a project plan; that is, specify how the defined goals should be achieved. (4) Execute the project according to the previously defined plan; that is, perform the planned development activities, manage the project, and collect measurement data. (5) Analyze project results; that is, compare hypotheses with real data and identify deviations and their reasons. (6) Package project experience; that is, capture project information in the project-specific experience base and update the organization-wide experience base (e.g., add a new effort baseline for a specific project context, update the significance of an existing model, or correct relevant models).
- The steps can be interpreted as follows for the whole organization: (1) Characterize the organization and identify trends. (2) Define general improvement goals and corresponding quantifiable hypotheses. (3) Choose pilot projects for validating the goals and hypotheses. (4)

Execute the pilot projects, collecting data regarding the goals and hypotheses. (5) Analyze the results and, in particular, validate the hypotheses. (6) Package project experience in the form of reusable experience elements and update/refine the existing experience base.

Fig. 8 illustrates the packaging step following project completion and focused on reusing and updating an effort model. The experience base, at the top of the figure, indicates the original state before packaging experience from a specific project, P. Model M1 has been used to forecast the effort distribution for this project. Model M1 is valid for projects of context C1 and has already been applied in S former projects. During analysis and packaging, the project results may lead to three different cases:

- Case 1: Model M1 was correct for project P; that is, the real effort distribution of project P was consistent with the distribution of model M1. In this case, the significance of model M1 is increased by one; that is, M1 has now been successfully applied in S + 1 projects.

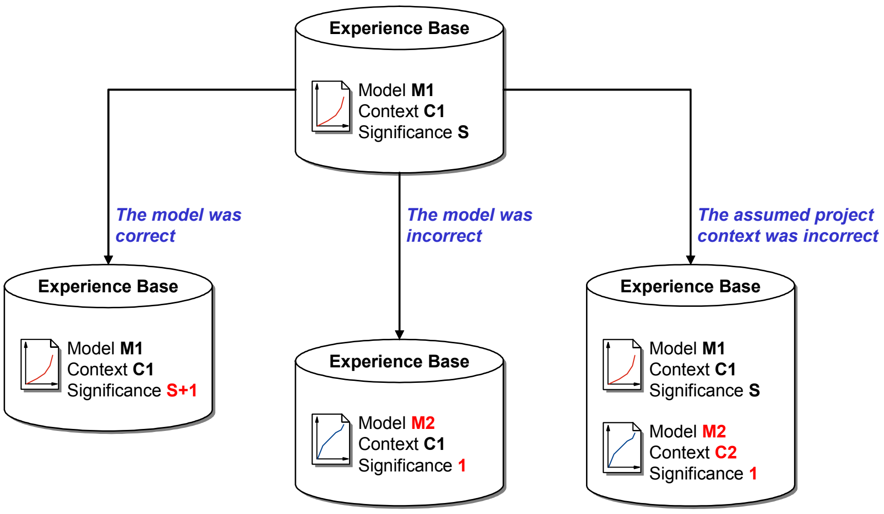


Fig. 8. Example for Packaging Project Experience.

- Case 2: Model M1 was incorrect for project P; that is, the real effort distribution of project P differed significantly from the effort distribution of model M1 (and no abnormal circumstances were recognized for project P). In this case, the original model has to be changed in order to reflect the new project experience. Model M1 is replaced by model M2 for context C1 and the significance of M2 is set to one, because M2 has been applied in only project.

- Case 3: The assumed context for project P was wrong; that is, a certain environment was expected (e.g., developer experience = `high`), but an analysis has shown that this original assumption was incorrect (e.g., the real developer experience = `low`). Therefore, the project context is actually a new context, C2. In this case, a new model M2 is added for context C2 with significance one and the original model is left unchanged for context C1.

5 Summary

This chapter describes how to capture, display, and use process information to support people performing a process. In addition, it provides many examples of specific capabilities that have proven valuable in practice.

First of all, we identified the needs for supporting people-oriented software development, discussed the relationships among roles, agents, and human process performers, and identified a variety of human characteristics and their influence on software engineering.

Then we provided an overview of different kinds of process information and how to collect this information prior to, during, and following process performance. We discussed the value of general, qualitative, status information about a process. In addition, we described one way to provide methodical, goal-oriented, support for collecting quantitative measurement data.

After that, we discussed a variety of gradually more significant ways in which people may be supported in carrying out their assigned tasks:

- First, we described simple support provided by process handbooks describing an agents' responsibilities, the activities they perform, the artifacts used and produced by the activities, resources supporting activity performance, and the conditions that reflect progress.
- After this, we discussed more extensive support provided by creating role-based workspaces that collect together all the information an

agent needs to access when carrying out a task or a set of inter-related tasks.

- Then we discussed even more extensive support that can be provided when the information displayed to agents not only reflects the process definition but also reflects, qualitatively and quantitatively, what has happened during process performance.

Subsequently, we presented extensions to this basic support which facilitates coordination and collaboration among agents cooperatively carrying out their tasks.

Finally, we discussed the opportunity to use the status information and measurement data to actively, automatically control process performance. In addition, we addressed how to use status information and measurement data not only to proactively support process performance (through enactment support) but also manage the experience gained during project performance.

Acknowledgments

Proof-of-concept work on specific and generic role-based workspaces was done by Mirjam Krementz[50]. The early work on EPGs was, in addition to the authors of the cited reference[16], influenced by Alan Christie, Denis Avrilionis and Anthony Earl. The PMC process documentation generation capability was, in addition to the authors of the cited reference[18], influenced by Dave Barstow, Bill Cohagan, Michael Mahan, Don Oxley and Dick Phillips. The following people participated in the development of SPEARMINT®/EPG: Danilo Assmann, Ulrike Becker-Kornstaedt, Fabio Bella, Dirk Hamann, Ralf Kempkens, Holger Neu, Alexis Ocampo, Peter Rösch, Louise Scott, Martin Soto, Martin Verlage, Richard Webby, and Jörg Zettel. The following students contributed to the implementation of SPEARMINT®/EPG: Martin Denn, Ralf Hettesheimer, Thomas Kiesgen, Dietmar Klein, Arne Könnecker, Dirk Ludwig, Sascha Schwarz, Slavomir Sobanja, Martin Vogt, and Christian Voigtländer.

References

1. B. Curtis, M.I. Kellner, and J. Over. *Process modeling*. Communications of the ACM, 35(9), pp. 75-90 (1992).
2. H.D. Rombach and M. Verlage. *Directions in software process research*. In Marvin V. Zelkowitz, editor, Advances in Computers, vol. 41, pp. 1-63. Academic Press (1995).
3. A. Endres and D. Rombach. *A Handbook of Software and Systems Engineering – Empirical Observations, Laws, and Theories*. Pearson Education Limited, Addison Wesley (2003).
4. IEEE 1058.1 — *Software Project Management Plans*. IEEE Computer Society Press, Los Alamitos, California (1987).
5. M. Paulk, B. Curtis, M. Chrissis, and C. Weber. *Capability Maturity Model for Software (V1.1)* (CMU/SEI-93-TR-024, ADA 263403). Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA (1993).
6. Pragma Systems Corporation, Reston, Virginia 20190, USA. [*http://www.pragmasystems.com*]
7. Integrated Process Asset Library, Federal Aviation Administration (FAA), Washington, D.C., USA. [*http://www.faa.gov/ipg/pimat/ipal*]
8. W. Scacchi. *Process Models in Software Engineering*. In: J. Marciniak (ed.), Encyclopedia of Software Engineering, 2nd Edition, Wiley (2001).
9. U. Becker, D. Hamann, and M. Verlage. *Descriptive Modeling of Software Processes*. In Proceedings of the Third Conference on Software Process Improvement (SPI '97), Barcelona, Spain (1997).
10. M. Verlage. *About views for modeling software processes in a role-specific manner*. In Proceedings of the second international software architecture workshop (ISAW-2) and international workshop on multiple perspectives in software development (Viewpoints '96) on SIGSOFT '96 workshops, San Francisco, California, United States, pp. 280-284 (1996).
11. R.W. Selby. *Amadeus Measurement-Driven Analysis and Feedback System*. In Proceedings of the DARPA Software Technology Conference, Los Angeles, CA (1992).
12. R. van Solingen and E. Berghout. *The Goal/Question/Metric Method - A Practical Guide for Quality Improvement of Software Development*. McGraw-Hill International (UK) (1999).
13. V.R. Basili, D.M. Weiss. *A Methodology for Collecting Valid Software Engineering Data*. In IEEE Transactions on Software Engineering, Vol. SE-10, No. 6, pp. 728-738 (1984).
14. V.R. Basili, G. Caldiera, and H.D. Rombach. *Goal Question Metric Paradigm*. In: "Encyclopedia of Software Engineering", Volume 1, edited by John J. Marciniak, John Wiley & Sons, pp. 528-532 (1994).

15. L.C. Briand, C.M. Differding, and H.D. Rombach. *Practical guidelines for measurement-based process improvement*. Software Process: Improvement and Practice, 2 (4) (1997).
16. M. Kellner, U. Becker-Kornstaedt, W. Riddle, J. Tomal, M. Verlage. *Process Guides: Effective Guidance for Process Participants*. Proceedings of the Fifth International Conference on the Software Process: Computer Supported Organizational Work, Chicago, Illinois, pp. 11-25 (1998).
17. U. Becker-Kornstaedt, J. Münch, H. Neu, A. Ocampo, and J. Zettel. *SPEARMINT® 6. User Manua*l. Fraunhofer Institute for Experimental Software Engineering, Report 031.02/E, Kaiserslautern (2003).
18. W. Riddle and H. Schneider. *Coping with Process Agility*. Tutorial at 2002 Software Engineering Process Group Conference (SEPG 2002), Phoenix, Arizona (2002).
19. W. Riddle. *Coping with Process Specification*. In Proceedings of Integrated Design and Process Technology (IDPT-2003) Conference, Austin, Texas (2003).
20. S. Miller and W. Riddle. *Experience Defining the Performance Management Key Process Area of the People CMM ... and the Link to Business Strategy*. In Proceedings of 2002 Software Engineering Process Group Conference (SEPG 2000), Seattle, Washington, (2000).
21. A. Ocampo, D. Boggio, J. Münch, G. Palladino. *Towards a Reference Process for Wireless Internet Services*. IEEE Transactions on Software Engineering, Special Issue on Wireless Internet Software Engineering, 29 (12), pp. 1122-1134 (2003).
22. F. Bella, J. Münch, A. Ocampo. *Capturing Experience from Wireless Internet Services Development*. In Proceedings of the International Conference on Applications and Emerging Trends in Software Engineering Practice (STEP 2003), Workshop on "Where's the evidence? The role of empirical practices in Software Engineering", Amsterdam, The Netherlands, pp. 33-39 (2003).
23. J. Zettel, F. Maurer, J. Münch, L. Wong. *LIPE: A Lightweight Process for E-Business Startup Companies Based on Extreme Programming*. Lecture Notes in Computer Science 2188, (Frank Bomarius and Seija Komi-Sirviö, eds.), Springer-Verlag, pp. 255-270 (2001).
24. J. Münch and J. Heidrich. *Software Project Control Centers: Concepts and Approaches*. Journal of Systems and Software, 70 (1), pp. 3-19 (2003).
25. W.W. Gibbs. *Software's Chronic Crisis*. Scientific American, pp. 86-95 (1994).
26. M. Shaw. *Prospects for an Engineering Discipline of Software*. IEEE Software 7(6), pp. 15-24 (1990).
27. H.D. Rombach and M. Verlage. *Directions in Software Process Research*. Advances in Computers 41, pp. 1-63 (1995).

28. Project Management Institute. *A Guide to the Project Management Body of Knowledge (PMBOK® Guide) 2000 Edition*. Project Management Institute, Four Campus Boulevard, Newtown Square, PA 19073-3299 USA (2000).
29. R. Hendrick, D. Kistler, and J. Valett. *Software Management Environment (SME)— Concepts and Architecture (Revision 1)*; NASA Goddard Space Flight Center Code 551, Software Engineering Laboratory Series Report SEL-89-103, Greenbelt, MD, USA (1992).
30. R. Hendrick, D. Kistler, and J. Valett. *Software Management Environment (SME)—Components and Algorithms*; NASA Goddard Space Flight Center, Software Engineering Laboratory Series Report SEL-94-001, Greenbelt, MD, USA (1994).
31. L. Landis, F. McGarry, S. Waligora, R. Pajerski, M. Stark, R. Kester, T. McDermott, and J. Miller. *Managers Handbook for Software Development—Revision 1*; NASA Goddard Space Flight Center Code 552, Software Engineering Laboratory Series Report SEL-84-101, Greenbelt, MD, USA (1990).
32. F. McGarry, R. Pajerski, G. Page, S. Waligora, V.R. Basili, and M.V. Zelkowitz. *An Overview of the Software Engineering Laboratory*; Software Engineering Laboratory Series Report SEL-94-005, Greenbelt, MD, USA (1994).
33. J. Heidrich and M. Soto. *Using Measurement Data for Project Control*. In Proceedings of the Second International Symposium on Empirical Software Engineering (Vol. II), Rome, pp. 9-10 (2003).
34. J. Heidrich. *Effective Data Interpretation and Presentation in Software Projects*. Technical Report 05/2003, Sonderforschungsbereich 501, University of Kaiserslautern (2003).
35. V.R. Basili, G. Caldiera, and H.D. Rombach. *The Experience Factory*. Encyclopedia of Software Engineering 1, pp. 469-476 (1994).
36. J. Grundin, S. Poltrock, and J. Patterson. *CSCW Overview*. Special Presentation at ACM Conference on Computer-Supported Cooperative Work (CSCW'96), Boston, USA (1996).
37. NetMeeting, Microsoft Corporation, Redmond, Washington 98052-6399 USA. [*http://www.microsoft.com/windows/netmeeting*]
38. Teamware Pl@za, Teamware Group Oy, Helsinki, Finland. [*http://www.teamware.net/Resource.phx/twplaza/index.htx*]
39. V. Basili et al. *Building an Experience Base for Software Engineering: A report on the first CeBASE eWorkshop*. In Proceedings of the 3rd International Conference on Product Focused Software Process Improvement (PROFES 2001), pp. 110-125, Kaiserslautern, Germany (2001).
40. W. Appelt. *WWW Based Collaboration with the BSCW System*. In Proceedings of SOFSEM'99, Springer Lecture Notes in Computer Science 1725, Milovy, Czech Republic, pp. 66-78 (1999). [*http://bscw.fit.fraunhofer.de*]

41. N.E. Fenton and S.L. Pfleeger. *Software Metrics. A Rigorous and Practical Approach*. International Thomson Computer: 2nd edition, London, UK (1996).
42. ACQWeb. Earned Value Management. [*http://www.acq.osd.mil/pm*]
43. J. Münch, J. Heidrich, and A. Daskowska. *A Practical Way to Use Clustering and Context Knowledge for Software Project Planning*. In Proceedings of the 15th International Conference on Software Engineering and Knowledge Engineering (SEKE 2003), pp. 377-384, San Francisco, USA (2003).
44. J. Münch and J. Heidrich. *Using Cluster Curves to Control Software Development Projects*. In Proceedings of the First International Symposium on Empirical Software Engineering (Vol. II), Nara, pp. 13-14 (2002).
45. J. Münch and J. Heidrich. *Context-driven Software Project Estimation*. In Proceedings of the Second International Symposium on Empirical Software Engineering (Vol. II), Rome, pp. 15-16 (2003).
46. C.W. Doerflinger and V.R. Basili. *Monitoring Software Development Through Dynamic Variables*. In Proceedings of IEEE Conference on Computer Software and Applications (COMPSAC*)*, pp. 434-445 (1983).
47. G. Dinkhoff, V. Gruhn, A. Saalmann, M. Zielonka. *Business Process Modeling in the Workflow Management Environment LEU*. In Proceedings of the 13th International Conference on the Entity-Relationship Approach (Lecture Notes in Computer Science, Vol. 881, pp. 46-63). Berlin, Heidelberg, New York: Springer (1994).
48. Christer Fernström. *ProcessWEAVER: Adding process support to UNI*X. In Proceedings of the Second International Conference on the Software Process: Continuous Software Process Improvement, Berlin, Germany, pp. 12-26 (1993).
49. R. Basili. *Quantitative Evaluation of Software Engineering Methodology*. Proceedings of the First Pan Pacific Computer Conference, Melbourne, Australia (1985).
50. M. Krementz. *Personal Workspaces for Electronic Process Guide (EPG) User*s. Project Thesis, Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern, Germany (1999).

Index