Software Project Control Centers: Concepts and Approaches

JÜRGEN MÜNCH\textsuperscript{a}, JENS HEIDRICH\textsuperscript{b}

\textsuperscript{a}Fraunhofer Institute Experimental Software Engineering, Sauerwiesen 6, 67661 Kaiserslautern, Germany, E-mail: muench@iese.fhg.de
\textsuperscript{b}Department of Computer Science, University of Kaiserslautern, PO Box 3049, 67653 Kaiserslautern, Germany, E-mail: heidrich@informatik.uni-kl.de

Abstract. On-line interpretation and visualization of project data are gaining increasing importance on the long road towards predictable and controllable software project execution. In the context of software development, only few techniques exist for supporting these tasks. This is caused particularly by the often insufficient use of engineering principles in the software development domain. Beyond that, interpretation and visualization techniques from other domains (such as business or production processes) are not directly applicable to software processes because of the specific characteristics of software development. A software project control center (SPCC) is a means for collecting, interpreting, and visualizing measurement data in order to provide purpose- and role-oriented information to all involved parties (e.g., project manager, quality assurer) during the execution of a project. This article presents a reference model for concepts and definitions around SPCCs. Based on this reference model, a characterization and classification of essential approaches contributing to this field is given. Finally, an outline for future research is derived from identified deficiencies of existing approaches.

Keywords. Software project control center, project controlling, quality assurance, data interpretation, data visualization.

1 Introduction

Software practitioners are faced with the challenging task of managing software development and maintenance processes in a predictable and controllable way. This requires particularly accurate and precise progress and quality monitoring. Techniques proven and routine in other engineering disciplines are considered radical innovations in software engineering (Brooks, 1995). Many software development organizations still lack support for obtaining intellectual control over their software development processes and for determining the performance of their development processes and the quality of the produced products. Systematic support for detecting and reacting on critical project states in order to achieve planned goals is usually missing. Practitioners still ask questions of the following type: Do we know how well our software development processes perform? What is the current project state? How can we determine whether we will reach the project goals on time? Who will alarm me early in case of critical project situations? How should I react in such situations? What are the weak points of our processes and what can we do to improve our processes? What are the benefits of improvement activities?

NOTICE: this is the author's version of a work that was accepted for publication in Journal of Systems and Software. Changes resulting from the publishing process, such as peer review, editing, corrections, structural formatting, and other quality control mechanisms may not be reflected in this document. Changes may have been made to this work since it was submitted for publication. A definitive version was subsequently published in Journal of Systems and Software, Vol. 70, Issue 1-2, (2004), pp. 3-19, DOI: 10.1016/S0164-1212(02)00138-3, http://www.sciencedirect.com/science/article/pii/S0164121202001383
There are several ways towards solving this problem. A widely accepted idea in the software engineering community is to introduce more engineering discipline into software development and maintenance (Shaw, 1990; Gibbs, 1994). This led, on the one hand, to the explicit definition of process models, product models, resource models, and quality models and their integration into comprehensive project plans (Rombach and Verlage, 1995). On the other hand, collecting and using measurement data may further enhance intellectual control of software processes and is a prerequisite for empirical model building. The establishment of measurement and feedback mechanisms helps to understand and control software development processes and products, and the relationships between them. It helps to make intelligent decisions and improve over time (Basili, 1996). On-line measurement (i.e., measurement during project execution) and feedback is of high importance because the quality of the final product is strongly dependent on the qualities of intermediate products and their respective creation processes.

One means to institutionalize measurement on the basis of explicit models (i.e., process models, product models, resource models, and quality models) is the development and establishment of so-called software project control centers (SPCC) for systematic quality assurance and management support.

An SPCC is comparable to a control room, which is a well known term in the mechanical production domain. It is used there as a central node for all incoming information of a production process. A control room collects all incoming data (e.g., the current state of an assembly line) and visualizes them for controlling purposes. If there is a problem with the production process (e.g., a blocked assembly line), the user of the control room is informed and can handle the problem. Another analogy is an air traffic control system that ensures the safe operation of commercial and private aircraft. Air traffic controllers use this system 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).

In this article we define a software project control center (SPCC) as a means for process-accompanying interpretation and visualization of measurement data:

It consists of (1) underlying techniques and methods to control software development projects and additional rules to select and combine them, (2) a logical architecture that clearly defines logical interfaces to its environment, and (3) a supporting tool that implements (parts of) the logical architecture.

Its input information includes but is not limited to information about project goals and characteristics, project plan information (e.g., target values for effort per development phase), measurement data of the current project, and empirical data from previous projects.

Its output information includes a context-, purpose-, and role-oriented visualization of collected and interpreted measurement data. That is, the visualization depends upon the context of the project, the purpose of the usage (e.g., monitoring), and the role of the user (e.g., project manager).

Its tasks include collecting, interpreting, and visualizing measurement data in order to provide context-, purpose-, and role-oriented information for all parties involved (e.g., project managers, quality assurer, developers) during the execution of a software development project. This includes, for instance, monitoring defect profiles, detecting abnormal effort deviations, cost estimation, and cause analysis of plan deviations.

After the introduction of the first measurement frameworks in the context of software development by Rubey and Hartwick (1968), Boehm (Boehm et al., 1978), Murine (1980), and Basili (1981), a large body of contributions in the form
of methods, techniques, tools, models, software development environments (SDEs), and software engineering environments (SEEs) has been proposed. However, most of the published work does not provide a focused view on on-line data interpretation and visualization for involved parties in order to provide them with relevant and empirically funded information for performing their tasks effectively. This may be the reason why practitioners, who want to set up and run an SPCC (as a kind of addendum to an SDE or SEE), are confronted with questions like: What are the main differences between existing approaches? Which approach is suited for my project or organization? Can I use an SPCC designed for a domain other than software development (e.g., a business performance management tool)? How can we package, store, and effectively use our past experience in such an SPCC? What are the consequences and effects of the use of existing approaches? What are deficiencies of existing approaches? Present answers to these questions are only insufficient. There is a lack of empirically-based practical guidelines on how to select and implement existing approaches into software development organizations. This might be a reason that hinders the practical application of SPCCs in ongoing development projects, aggravates management, and contributes to time and money expenditures as well as quality problems.

Attempting to contribute to a solution of the problem, the goal of this article is to give a comparative overview of published approaches suited for project control and guidance and demonstrate their contribution to the concepts of an SPCC. The article is organized as follows: Section 2 provides the basic SPCC terminology. The description is organized around five dimensions—purpose, technical, improvement, role, tool—with which we focus on key aspects of SPCCs. Section 3 discusses the operational SPCC concept by sketching an improvement-oriented software engineering model, which serves as a basis for further descriptions. Furthermore, we introduce a two-dimensional characterization schema and suggest a logical architecture for SPCCs. In Section 4, essential approaches are described and classified along our characterization schema. Finally, Section 5 summarizes the article and gives an outlook on future research directions in this field.

## 2 SPCC Terminology

The following section provides basic definitions, which form a consistent terminology and are useful for understanding the description of existing approaches in Section 4. A software project control center (SPCC) can be classified according to five dimensions: purpose, technical, improvement, role, and tool dimension. Each dimension consists of several elementary characteristics (see Figure 1). The following subsections focus on these dimensions and give a short definition for their characteristics.

### 2.1 Purpose Dimension

The purpose dimension describes the possible usage purposes of an SPCC (such as monitoring or prediction) and therewith the basic (external) functionality. The following terms do not give a complete list of possible SPCC usage purposes, but aim at defining an extensible and adaptable basic structure. Extensibility means the capability to introduce a new usage purpose together with associated techniques and methods that implement the purpose. Adaptability means the capability to choose different techniques and methods to implement a certain purpose (or a group of purposes) and the possibility to tailor them to the context of a specific
software development project. Techniques and methods (for different purposes) form the basis of the SPCC functionality. Therefore, we will refer to them as SPCC functions in the following: An SPCC function is a technique or method that implements a certain purpose or a group of purposes. The latter is the case if a function covers more than one single purpose.

The following definitions are adapted from the Software Management Environment (SME) approach (Hendrick et al., 1992), which was developed at the NASA/SEL.

**Monitoring** refers to observing the project state and progress by surveying attributes or combinations of attributes from processes, products, and resources of the project. Briand et al. (1996) define monitoring as following the trends/evolution of the performance/state of processes and products.

**Comparison** aims at using archived data from completed projects or nominal performance guidelines as references to judge the progress and health of the current project.

**Analysis** aims at (1) examining the monitoring results and (2) applying information about the project context to identify the probable causes of deviations from the nominal performance guidelines.

**Assessment** aims at weighting information about the project to form a judgment of project, product, and process quality.

**Prediction** aims at extrapolating attributes of processes, products, and resources of the project from the current project status towards project completion to assess the future behavior of the project. In general, prediction always requires some kind of mathematical model. Fenton and Pfleeger (1996) define prediction as identifying relationships between various process and product factors and using these relationships to predict relevant external attributes of products and processes. One means for capturing the dynamic behavior of development projects can be

---

1In the SME approach the function is called observation.
simulation modeling, i.e., values for key parameters can be forecasted with simulation. This requires valid simulation models for the specific project contexts.

Planning, in the context of an SPCC, aims at defining baselines or nominal values for certain measures. In addition, it aims at assessing (alternative) planning decisions and their effects. This is the basis for further dynamic replanning during the execution of the project.

Guidance aims at proposing 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 might use measurement-based guidance as assistance for harmonizing his own performance with the overall process and given project goals. Lott (1993, 1994) discusses several approaches in the context of measurement-based guidance systems. Lott and Rombach (1993) investigate the possibilities of providing measurement-based guidance using explicit project plans.

Other purposes: (a) Control consists of influencing the course of a project in order to alleviate risks (Briand et al., 1996). (b) Learning implies modifying the process from project to project in order to improve quality or productivity (Briand et al., 1996). (c) Certification is a procedure by which a third party gives written assurance that a product, process, or service conforms to specified requirements (ISO/IEC, 1996). (d) Benchmarking refers to measuring and comparing an organization’s process/product against processes/products from other companies and other internal sources (Harrison et al., 1997).

Some purposes are based on others: For instance, comparison requires monitoring to get data about the progress and health of the current project, guidance needs analysis to identify deviations from the nominal performance guidelines. The detailed relationships between all purposes depend upon the functions that implement the corresponding purposes; that is, they depend upon the techniques or methods used.

2.2 Technical Dimension

The technical dimension addresses the internal procedures, which are essential for providing the external functionality of an SPCC.

Goal definition aims at defining the goals for applying an SPCC. One approach for the systematic definition of measurement goals in the software domain is the goal question metric paradigm (GQM) (Basili and Weiss, 1984; Basili and Rombach, 1988; Rombach, 1991; Basili, 1992; Basili et al., 1994b; Briand et al., 1996; van Solingen and Berghout, 1999). The goal definition of this approach can be adapted to our context. A measurement goal consists of the object of study (e.g., code modules), the purpose of the usage (e.g., monitoring), the quality focus to be analyzed (e.g., defects), the viewpoint/role of the user (e.g., quality assurer), and the project context (e.g., project A in company X). According to (Briand et al., 1996) each goal should not cluster more than one purpose, quality focus or viewpoint. It is possible that an SPCC supports several goals at the same time.

Derivation of measures aims at goal-oriented gathering of appropriate measures. For each goal the important attributes to be measured have to be derived. This can be done by using the GQM paradigm and involving the relevant personnel (e.g., project members, management staff).

Data collection aims at goal-oriented collection of process, product, and resource (personnel) data of interest, concurrently with software development (Basili and Weiss, 1984). Giese et al. (1994) derive requirements for data
collection concerning technical and nontechnical issues. Data validation is performed concurrently with software development and data collection to ensure accuracy of data (Basili and Weiss, 1984). Particularly, this is important with regard to subjective data, which can vary depending on the person collecting the data.

Data processing is defined as the systematic performance of operations upon data such as handling, merging, sorting, and computing. In the context of an SPCC, data processing denotes all operations necessary for executing the functions that implement SPCC purposes.

Presentation and visualization is the process of representing abstract business or scientific data as images that can aid in understanding the meaning of the data. For clarity reasons, we distinguish the terms presentation and visualization: Presentation is the process of prepackaging processed data that is to be visualized, and accordingly, visualizing is the process of building an image of data presented. Presentation and visualization can take place on different organizational levels (such as project level, multi-project level, organizational unit level, or company level).

Model building aims at generating an abstraction of the real world. Within an SPCC, model building is particularly needed to provide a basis for the comparison of current software development projects with models of previous ones, respectively with nominal performance guidelines. Generally, we distinguish between process models, product models, and quality models. A process model describes a class of activities or actions with common characteristics (Basili and Rombach, 1991). A product model describes a class of concrete documents or artefacts with common characteristics (Basili and Rombach, 1991). A quality model describes certain quality aspects and their relationships for the purpose of characterization, prediction, and evaluation. For instance, a quality model describes the relation between test effort and design complexity (Feldmann et al., 1998). In the context of an SPCC, a process model could be used to synchronize data collection procedures with development activities. Predictive quality models could be used to define target values and thresholds.

2.3 Improvement Dimension

The improvement dimension addresses the underlying improvement or learning approach of an organization that uses an SPCC. We propose a quality-oriented improvement approach, the so-called quality improvement paradigm (QIP), and a supporting organizational concept to institutionalize the collective learning of an organization, the so-called experience factory (EF) (Basili and Caldiera, 1991; Basili et al., 1994a).

The project-specific improvement cycle aims at measuring relevant project data during project execution to improve project control and provide real-time feedback. The SPCC acts as a means for process-accompanying data visualization and interpretation to support project control.

The strategic improvement cycle aims at packaging experience (e.g., in the form of improved models) for future projects. The information provided by an SPCC can be used to describe experience more precisely and thus improve packaging.

The experience base stores experience used by projects currently running as well as packaged experience from previous projects. It consists of two sections: The project-specific section of an experience base can be defined as the union of the project databases of all performed projects of interest. This includes (especially
2 SPCC TERMINOLOGY

in the context of an SPCC) measurement data of all respective projects. The organization-wide section of an experience base consists of packaged experience in the form of reusable units (e.g., adaptable process models, product models, predictive quality models, lessons learned) together with information about their scope of applicability and their empirical validation.

An improvement paradigm is a paradigm for ongoing improvement of product and/or process quality. Well-known approaches include the plan-do-check-act (PDCA) cycle (Deming, 1986), the total quality management (TQM) for software development (Zultner, 1993), and the quality improvement paradigm (QIP) (Basili et al., 1994a). The QIP is an iterative, goal-driven framework, which includes steps for goal setting, planning, executing, and evaluating. It considers the collection of measurement data in order to support project execution (e.g., make quality assurance possible) and to construct new or improved models. The QIP is the basis of the improvement-oriented TAME model (Basili and Rombach, 1988).

2.4 Role Dimension

The role dimension addresses the potential users of an SPCC. In the following, we define a selection of important roles that have their own view onto the data of an SPCC. The definitions are partly adapted from Rombach and Verlage (1995).

A project planner creates a project plan, starting from project goals and characteristics, and adjusts it during project execution. An SPCC could support this role by pointing out plan deviations and providing alternative plan modifications along with appropriate information for decision making about these alternatives.

A project manager ensures that the project is executed according to the plan. An SPCC could support this role by providing current project data (e.g., percent cost expected, estimated cost to complete, resource allocation), indicating project health and recommending corrective actions, if necessary.

A quality assurer controls deliverables and processes with respect to the quality goals specified in the project plan to provide confidence in their fulfillment. An SPCC could support a quality assurer in collecting, validating, presenting and visualizing all quality-related data that is important with respect to defined quality goals (e.g., defect distributions among software modules). This might include the definition of thresholds and the implementation of early warning mechanisms for the detection of actual or forecasted goal violations.

A quality manager manages knowledge and experience across projects, provides it to current projects during planning and execution, and improves it on the basis of project knowledge. An SPCC could support this role by providing project data for packaging experience (e.g., building new models, building more precise models, creating model variants, rejecting models). For instance, an SPCC could provide a project trace, which allows the quality manager to analyze the situation and the project context in which experience was gained. This could improve packaging (e.g., experience gained in unusual situations could be treated differently than experience gained during intended project performance). The other way round, a quality manager could provide models as input for an SPCC (e.g., measurement plans for organizing data collection, estimation models for defining target values).

A developer produces and modifies artefacts from system requirements to component code based on previous artefacts (such as specifications) or implicit knowledge (such as customer ideas) according to project guidelines. An SPCC could provide a developer with feedback that guides his performance. This might
include helping a developer to realize that he is introducing inconsistencies into the project (e.g., warning of threshold overruns, prohibiting the check-in of artefacts that violate specifications) and helping him to be informed on which process is enabled or which products are available (by means of visualization of process and product states).

Other roles can be supported with an SPCC. For instance, a tester might be interested in data about test coverage or a configuration manager could use an SPCC to monitor product states and their transitions.

2.5 Tool Dimension

The tool dimension addresses aspects relevant for building and implementing an SPCC. Distributed development may influence the construction of an SPCC in different ways. First, distribution of software development to different locations requires mechanisms to cope with different formats, heterogeneous platforms, network technology, etc. Second, distributed control of software development activities demands mature coordination and cooperation mechanisms. Besides that, building an SPCC for distributed development has to consider cultural differences, varying expectations, different privacy policies, etc.

The architecture of an SPCC can be separated into a logical architecture and a physical architecture. The logical architecture describes an SPCC as a composition of logical components (e.g., visualization component, model repository) and their interfaces. The physical architecture specifies the implementation-oriented design and describes physical components (such as database handler, installer) and their connections. Logical and physical architecture may differ, but a defined mapping between their parts should exist.

Data management aims at (1) specifying data elements with heterogeneous formats (e.g., adding statements about the type, scale, interval, and host), (2) collecting data according to the specification, (3) processing the collected data according to the functionality of an SPCC, (4) updating collected and processed data in case of dynamic replanning, and (5) storing data appropriately.

Compatibility aims at (1) providing interfaces for an SPCC to different development and management tools (such as unit test tools or process engines), (2) adapting connections to project-specific and organization-wide experience bases, and (3) integrating an SPCC dynamically into organizational strategies.

Other aspects: There are some more aspects that are relevant for building and implementing an SPCC: (a) Tailoring aims at mechanisms to adapt an SPCC to organizational requirements and project-specific characteristics (e.g., defining new functions, selecting views). An SPCC should provide mechanisms for coping with variants of project contexts, i.e., the tailoring of functions, views, data interfaces etc. should be supported. Important issues concern understanding of relevant variation parameters, appropriate representation of generic functions and views, the development of tailoring mechanisms, and the packaging of functions and views for reuse in further projects. (b) Portability aims at using the SPCC on a hardware and software platform other than the one on which it was created (without requiring major rework).

3 Operational SPCC Concept

The previous section introduced basic SPCC definitions according to five
dimensions but did not address how to set up and run an SPCC that is able to take these dimensions into account. This section provides information about the operational concepts of an SPCC, that is, how we can integrate an SPCC into a project environment, what basic requirements for an SPCC are, and finally, what a possible logical architecture looks like.

3.1 Software Development Model

Experience from analyzing software development projects, their processes, products and organizational issues has led to several proposals on how to organize software organizations, how to perform projects in an engineering style, and how to learn from past projects. We do not want to present here a new framework comprising all these issues, but rather build on the extensive body of knowledge that is incorporated in the available proposed approaches. We concentrate on data interpretation and visualization during the execution phase of a project in order to provide relevant and empirically funded information for performing tasks of the execution phase more effectively. Our focus is on the relevant aspects for software project control centers in an established software development model.

We use the TAME (Tailoring a measurement environment) software development model (Basili and Rombach, 1988, 1991) as a basis for the integration of an SPCC into an SEE. One main characteristic of this model is the assumption that software engineering processes need to be tailorable and tractable on the basis of measurement. TAME provides a suitable environment for integrating project control centers because it makes all necessary information available and comprises essential mechanisms for capturing and using software engineering experience. The TAME model has been a major source for the development of engineering-style methods like the goal question metric paradigm (GQM), the quality improvement paradigm (QIP), and the experience factory organization (EF). The model has been instantiated in several progressive software engineering environments, such as NASA-SEL (McGarry et al., 1994) or MVP-E (Becker et al., 1997).

According to Basili and Rombach (1988), integrated SEEs must support mechanisms for on-line data collection from development processes and feedback of measurement results into the development process. Strong ties between development and measurement-oriented subsystems of a development organization are claimed. We propose an integrated software development model for SEEs based on these requirements and the principles of the TAME model. We distinguish four different levels that build upon each other: (1) roles, (2) services, (3) tools, and (4) information level (see Figure 2).

In the following we exemplarily assign several roles, services, tools and information units to each of the four levels in order to illustrate the integration of an SPCC within a software development project.

Roles. The roles involved in software development projects use different services in different phases of the project to fulfill their tasks. For instance, a project planner uses planning services in order to create a project plan, a developer uses technical development services in order to develop artefacts, a project manager uses management services in order to control the project, and so on.
Services. We assume that all services needed to conduct a software development project can be classified along three dimensions: planning services, execution services, and know-how management services. (1) Project planning is done based on explicit project goals and characteristics. During planning, models (e.g., process models, product models) are instantiated and related in order to build a project representation with respect to the project’s goals and characteristics. This includes the determination of quantitative target values based on experience from past projects. The project plan can be used for communication, coordination, resource assignment, and quality assurance purposes during project execution. We distinguish initial planning, which refers to planning before project start, from replanning, which addresses the systematic changing or detailing of the plan during project execution. We make this distinction because replanning requires additional input such as the current project state, which can be provided by an SPCC. (2) The services mainly needed for project execution can be divided into services for technical development in order to develop artefacts, services for project management in order to control the software development project itself, and, finally, services for quality assurance in order to assure the quality of resulting artefacts. The services provided by an SPCC mainly support the three project execution services by providing, for instance, information about developed artefacts, the current project state, or quality goals of the project. Therefore, the SPCC’s services can be seen as orthogonal project execution services. (3) Know-how management is used to analyze the collected project data in order to provide and improve existing models for future use. Generally, know-how management also includes generalizing or formalizing experience in the form of models, guidelines,
etc.

Tools. Some services are performed automatically or at least semi-automatically; that is, a human agent is supported by dedicated tools. For instance, there exist tools to support project planning, to measure project data, to develop artefacts, or to package experience. All these tools are invoked by services and use information resulting from other tools or from an experience base to perform their tasks. An SPCC tool is mainly deployed during the execution of a project. It supports roles such as project manager, quality assurer or developer (respectively corresponding services) by providing, processing, interpreting, and visualizing process-accompanying data for their specific needs and purposes. Measured on-line project data is retrieved from the project-specific section of an experience base. The experience base stores information currently used by the project as well as packaged experience from previous projects. Packaged experience from previous projects can also be used by an SPCC. For instance, quality models such as estimation models can be used for defining thresholds, or qualitative experience such as lessons learned can be used for replanning. This kind of experience is stored in the organization-wide section of the experience base.

Information. The fourth level of our development model addresses the information needed to perform a software development project; that is, information needed directly by services or needed by tools to support respective services. We assume that during project execution, measurement data is collected and validated. One task of the information level is to provide project-specific or organization-wide information during project execution in order to improve control over the project. This information is stored in a certain kind of know-how repository. For this, we propose an experience base. The project-specific section includes, for instance, measurement plans and data, project plans and traces, and products (or artefacts). The organization-wide section includes models (e.g., process models, product models, and quality models) and corresponding objects (e.g., products produced and process traces).

3.2 Characterization Schema

Within an information processing system, one can roughly distinguish between three main tasks: collection and validation of input data, processing and transformation of the data within the system, and finally, presentation and visualization of the results of the data processing step. There exist a lot of frameworks/paradigms for collecting and validating data in a goal-oriented way (like the already mentioned goal-question-metric paradigm). An SPCC is based on this data but is mainly responsible for (a) processing and (b) presenting and visualizing the data in a goal-oriented way. Therefore, we want to describe the quality of an SPCC along these two dimensions. Several other dimensions are conceivable, but we proclaim those two as very important in order to achieve goal-oriented SPCC usage.

The first dimension characterizes the kind of data processing, particularly the capabilities to process data in a generic way. We distinguish between:

- **Predefined functions.** This characteristic refers to the existence of a predefined set of functions, which implement a certain usage purpose or a group of purposes. The functions are processed during a project without context adaptations. Moreover, the integration of new functions (e.g., a new analysis or prediction technique) is not possible or at least very difficult.
- **Predefined functions and context-oriented adaptation.** The second characteristic
adds context-oriented adaptation; that is, in addition to having a static set of predefined functions, all functions can be adapted according to a certain project context. For instance, the kind of analysis technique (like cluster analysis) is fixed, but the function itself can be adapted according to a certain project context.

Variable functions and context-oriented adaptation. This characteristic enhances the capability of an SPCC to use variable functions; that is, now the set of functions itself is expandable. For instance, if a project manager wants to apply a certain technique to estimate project costs, he is able to expand the set of estimation functions with regard to the new technique. That is, an SPCC usage purpose can be covered by more than one function and an SPCC user can choose and adapt the appropriate one to implement a certain usage purpose.

The second dimension characterizes the kind of presentation and visualization that is used. We distinguish between:

Static presentation and visualization. This signifies that the kind of data presentation and visualization of an SPCC is the same for different usages within varying contexts. The role and purpose of an SPCC usage are fixed or not considered.

Purpose-oriented presentation and visualization. A purpose-oriented presentation allows to adopt data presentation and visualization according to the purpose of an SPCC usage, for instance, monitoring, comparison, analysis, or prediction. The purpose is implemented by underlying SPCC functions.

Purpose- and role-oriented presentation and visualization. This characteristic adds role-oriented presentation aspects; that is, data visualization and presentation is combined with a certain SPCC usage role. The data can be presented according to a certain viewpoint. For instance, a project manager may need a more abstract visualization of effort data than a developer.

In order to achieve optimal benefit from the usage of an SPCC, we proclaim an approach that allows variable functions and context-oriented adaptation as well as purpose- and role-oriented presentation and visualization.

3.3 Logical SPCC Architecture

Based on the derived characterization schema of the previous section, this section presents a basic architecture for an SPCC that is organized along three different layers. The information layer gathers all information that is essential for the SPCC functionality, for instance, measurement data of the current project, experiences of previous projects, and internal information, like all available purpose-oriented SPCC functions (see Section 2.1). The functional layer performs all data processing activities; that is, it performs the used SPCC functions and composes the role-oriented SPCC views (see Section 2.4). Finally, the application layer is responsible for all interactions with an SPCC user; that is, it provides the resulting information of the functional layer to an SPCC user and receives all incoming user requests.

Each layer consists of several conceptual units, which provide the essential functionality. In the following, we will briefly describe the tasks of every conceptual unit and relate them to the derived requirements. An overview of the software project control center’s architecture is presented in Figure 3.

Pool Management. Basically, we distinguish between three types of expandable, generic SPCC pools. The first one is the pool of functions which provides techniques and methods for several usage purposes, like monitoring, prediction, or guidance. In order to present the results of the functions according to
a certain user viewpoint, we need explicitly defined views of the processed data, such as one presentation suitable for the project manager, one for the quality assurer, and so on. This concept is represented by a pool of views. Finally, we have to visualize the views according to an output tool or device, like a web-browser, Gnuplot, or Microsoft Excel. Therefore, it is useful to provide a set of output interfaces that form the third and last information pool of an SPCC. Together, SPCC functions, SPCC views, and output devices form a three-layered visualization catena. SPCC functions process the measurement data, SPCC views present the resulting information according to a certain viewpoint, and output devices visualize the views according to a certain output tool or device. The pool management unit is responsible for the access to the three pools; that is, it retrieves appropriate functions, views, and output devices on the one hand, and on the other hand, it stores new, generalized functions, views, and output devices into the according pools.

**Figure 3.** Software project control center’s architecture.

*EB Management.* As already mentioned, we need access to a twofold experience base (EB). One section provides project-specific information, like the measurement data of the current project, the project goals and characteristics, and the project plan. The other section provides organization-wide information, like
quality models (e.g., as a basis for predicting measurement data) and qualitative experience (e.g., to guide project managers by providing a course of actions). The EB management unit organizes access to an experience base by providing mechanisms to access distributed data sources (in case of distributed development of software artifacts), validate incoming data, and integrate new experiences gathered through the usage of the SPCC into the (organization-wide) EB. Therewith, the EB management unit provides all information necessary in order to perform the chosen SPCC functions.

**Customization.** The customization unit is the most complex conceptual unit of an SPCC. (1) At first, we have to initialize the EB; that is, we have to define all data sources for SPCC usage. (2) Then, we need goal-oriented tailoring of the SPCC pools; that is, we need to choose appropriate SPCC functions, views, and output interfaces according to a certain usage or measurement goal. An SPCC has to integrate a methodology for a goal-oriented derivation of suitable functions, views, and output interfaces from a respective measurement plan (e.g., a GQM plan). (3) Thereafter, we need to adapt the resulting three-layered visualization catena according to the project goals and characteristics. (4) Finally, if new functions, views, or output interfaces are defined by an SPCC user, we need to generalize and integrate them into the respective pools for future SPCC usages.

**Data Processing.** The data processing unit receives the chosen and adapted SPCC functions from the customization unit. It analyzes all functions; that is, it determines input and output information, the function body, and the relationships with other functions. During execution of the chosen functions, the data processing unit receives the respective input information from the EB management unit, respectively from a previously executed function. The results of all executed SPCC functions are delivered to the presentation unit for a viewpoint-oriented presentation.

**Presentation.** The presentation unit receives the chosen and adapted SPCC views from the customization unit. In analogy to the data processing unit, it analyzes all views; that is, it determines the relationship between the views, and it determines which results of the SPCC functions have to be integrated. The results of all SPCC views are delivered to the user communication unit for device or tool-specific visualization.

**Packaging.** The packaging unit summarizes all experiences gathered through the usage of an SPCC, adapts them according to the needs of future projects (i.e., generalizes the information units) and delivers them to the EB management unit in order to integrate them into the respective section of an experience base.

**User Communication.** At last, we need a unit for communication with an SPCC user (for instance, via a web browser). This unit has the following tasks: (1) It determines the access granted to a specific SPCC user; that is, it permits a certain user to access the results of a certain set of functions, respectively a certain set of views. (2) It provides a graphical user interface (GUI) in order to customize the SPCC according to project goals and characteristics (via the customization unit). (3) It visualizes the views (delivered by the presentation unit) according to the chosen output interfaces.

The conceptual units above present a high level description of a logical SPCC architecture, which allows variable functions and context-oriented adaptation by the concept of appropriate pools for functions, views, and output interfaces, and which allows purpose- and role-oriented presentation and visualization of data by building up a so-called visualization catena for each SPCC user, respectively usage role.
4 Existing Approaches

The last section introduced the operational concept of an SPCC on the basis of the terminology of Section 2 and proclaimed a logical architecture. The goal of this section is to fill the SPCC concept with life; that is, to discuss some selected approaches in the SPCC field and show how these approaches or parts of them fit into our architecture.

4.1 Research Scope

The scope of an overview usually depends upon the direction of the research effort for which the overview is intended. Existing papers in the field typically reside on a specific level of abstraction and present the work from a specific perspective. There are papers describing frameworks (e.g., Lott, 1993), papers describing methods and techniques (e.g., Tesoriero and Zelkowitz, 1998a), and papers describing tools (e.g., Hendrick et al., 1992). Typical perspectives found are improvement perspectives (e.g., learning organization, measurement programs), mathematical perspectives (e.g., model building), or empirical perspectives (e.g., data analysis and validation).

Figure 4. Scope of the overview.

In this article, we define the scope in the following way: We focus on tool-supported approaches for on-line data interpretation and visualization in the context of software development (see Figure 4). We consider the selected approaches for the overview as representative of the work in this scope. For rendering the scope more precisely, we exclude several approaches:

First, we exclude approaches from other domains such as mechanical production processes (e.g., supervision of a coal power plant). The reason is that other domains have different characteristics and as a consequence, specific controlling approaches for these domains are not or not directly applicable for controlling software development. For instance, mechanical production processes are typically characterized by consistently identical production cycles, which are repeatedly performed in short time intervals. The small variance of mechanical production processes and the large quantity of data allows for completely different control and data analysis techniques and methods. In the area of business processes (e.g., accounting, acquisition) exist a lot of controlling approaches (e.g.,
performance management with user-configured dashboards). In contrast to software development processes, business processes are usually deterministic. However, approaches for controlling business processes are often used to control software development. For this reason, we picked up a typical approach from the business process domain for the overview and show the limitations for its use in the software development domain.

Second, we exclude solely resource-oriented approaches (such as attendance/absenteeism control, person-job matching, performance appraisal) and solely product-oriented approaches (such as simple configuration control). The rationale behind this is that the quality of a developed software product and the efficiency of the corresponding process depend upon more than one dimension. Therefore, we only take approaches into account that have a wider view. That means, we only selected control approaches that at least consider process and product aspects.

Third, we exclude spreadsheet programs (e.g., Microsoft Excel), diagramming programs (e.g., Microsoft Visio), and function plotting programs (e.g., gnuplot). Spreadsheet programs can be used as front-end of an SPCC for measurement collection and validation. The familiarity of developers and managers with such tools might facilitate data collection procedures and, as a consequence, increase the acceptance of measurement activities. Diagramming and function plotting programs can be used as front-end of an SPCC. They can be used to turn data into diagrams and provide a variety of specialized diagram types.

4.2 Discussion of the Approaches

In the following we will briefly describe 7 approaches and classify them according to the characterization schema of Section 3.2, namely the two dimensions (a) data processing and (b) data presentation and visualization.

Most of the following approaches reside in the software development domain (in detail, Provence, Amadeus, Ginger2, SME, WebME, and PAMPA). One approach originates from the domain of business processes (namely, PPM) and is discussed here because it can be considered as a typical representative of existing business-oriented controlling tools.

Figure 5 presents an overview of the approaches discussed and classifies them according to the two dimensions mentioned above. Each approach is listed with a respective publication and the connection to preferred or required process engines or software development environments (SDEs). If an approach is open with regard to this point, that is, if it can use different process engines or SDEs or is independent from a certain process engine or SDE, we term this open interface.

As we see in Figure 5, there is a lack of approaches that support purpose- and goal-oriented visualization and presentation and provide an expandable set of functions with context-oriented adaptations at the same time.

**Provence.** Provence (Krishnamurthy and Barghouti, 1993) is a framework for project management. It informs managers about state changes of processes and products, and is able to generate project reports. Furthermore, it allows project managers to initiate dynamic replanning steps. The main idea behind the Provence framework is an open and adaptable architecture. Most process-centered software development environments (SDEs) depend upon a monolithic structure; that is, they handle all tasks within the specific SDE. The component-based architecture of Provence allows the system to be flexible and facilitates the integration into
different organizations. Provence observes the development process, captures process and product data, answers queries about the current project state, and visualizes process transitions.

The prototypical instantiation of the Provence framework uses the rule-based SDE Marvel\(^2\) (Kaiser et al., 1990) as a process server and the programmable graph editor Dotty for visualization. There exists a set of predefined functions (such as for monitoring the software development process and analyzing the project state) and a static visualization of process transitions. Thus, the capabilities concerning purpose- and goal-oriented data presentation and visualization, and variable and context-oriented data processing, are limited.

![Figure 5. Comparison of existing approaches.](image)

**Amadeus.** The metric-based analysis and feedback system Amadeus (Porter and Selby, 1990; Selby et al., 1991) is embedded into the process-centered SDE Arcadia. The goal is to integrate measurement into software development processes and to establish analysis and feedback mechanisms by providing functions in order to interpret process events, object state changes, and calendar time abstractions. Amadeus is based on a script language, which dynamically interprets the three kinds of basic events that can be combined in order to create more complex ones. An Amadeus user defines (reusable) scripts to observe certain events. Events are kinds of triggers for user-dependent agents, which execute a number of actions, like collecting specific data items of the project. All collected data is analyzed by humans or automatically. For instance, a method called classification tree analysis (which will be described later) is used to classify components of a software system.

Amadeus uses active agents to interpret user-defined scripts. This approach

\(^2\)Marvel is not used in Provence to support further software development activities, due to its monolithic structure.
allows to add new functionality by creating a new script or expanding the corresponding tool kits. The data is presented according to a certain purpose of usage, like classification, interconnectivity, sensitivity, or descriptive analysis. However, there are no special concepts for providing role-oriented presentation and visualization (i.e., present the results from a certain viewpoint).

**Ginger2.** The Ginger2 system (Torii et al., 1999) implements an environment for computer-aided empirical software engineering (CAESE). Torii et al. present a framework that consists of three parts: (1) A life-cycle model for empirical studies, (2) a coherent view of experiments through data collection models, and (3) an architecture that forms the basis of a CAESE system. The main idea is to center the experimental aspects of software development. Within a CAESE system a software engineering problem is given, which consists of questions and hypotheses. The goal of the CAESE approach is to find knowledge about the given problem statement. The focus is more on conducting controlled experiments (so-called *in vitro* studies) and less on developing software products within a real software development project (so-called *in vivo* studies).

It fulfills its tasks with a multitude of predefined data collection and analysis techniques, for instance, gathering audio and video data of the experiment. The underlying concepts do not provide context-oriented adaptations of the data processing functions. The data is presented according to the needs analysis before the experimental design. There are no special concepts for providing role-oriented presentation and visualization, in spite of a goal-oriented design of the experiment itself.

**SME.** The software management environment (SME) (Hendrick et al., 1992, 1994) was developed at the software engineering laboratory (SEL) (Landis et al., 1990; McGarry et al., 1994) of the NASA Goddard Space Flight Center (GSFC). The SME is a tool to provide experience, gathered by the SEL, to project managers. The usage of the SME presumes that software development takes place within a well defined management environment. The implementation of SME functions relies on information from three separate databases: The SEL database includes information from previous projects; that is, subjective and objective process and product data, plans, and tool usages. The SEL research results database includes different models (such as growth or effort models) and relationships between certain parameters/attributes (described with quality models). Primarily, they are used to predict and assess attributes. Finally, the SEL management experience database includes the experience of managers in the form of rules within an expert system. They help inexperienced managers to analyze data and guide replanning activities. For instance, this database includes lists of errors and appropriate corrective actions.

The SME is an automated management tool, which provides a predefined set of techniques in order to observe, compare, predict, analyze, assess, plan, and control a software development project; that is, to support well-founded decision making. The functions are performed according to certain project characteristics within the SEL database. The presentation is purpose- and goal-oriented, even though the whole system is generally focused on the project manager.

**WebME.** The web measurement environment (WebME) (Tesoriero and Zelkowitz, 1997, 1998a) is a web-based data visualization tool that is based on the SME approach. WebME enhances the capabilities of the SME in terms of
distributed development of software. While the SME is concentrated on development within a certain closed SDE, WebME supports development at different locations with heterogeneous SDEs and provides appropriate data integration mechanisms. WebME uses a special script language (the so-called data definition language), which is able to integrate information of different heterogeneous SDEs into one common view.

The mediated architecture consists of the following three layers: The end-user applications layer provides access to the WebME system via a web browser. A user asks a query and gets the answer in the form of HTML pages. This makes the architecture platform independent and allows access from every location within the world wide web. The mediating information servers layer is the central processing layer. A query processor receives data from the web browser and transforms them to legal queries according to the WebME system. Vice versa, it transforms the answers of the WebME server to HTML pages and transmits them to the web browser. To know which data of which host in which format is to be used, the WebME system needs appropriate meta data in the form of scripts. Finally, the information resources include a data wrapper for each development location of a distributed project. The data wrapper receives data from a local database and transmits them to the WebME system.

Because distribution is no dimension according to our characterization schema, WebME and SME are classified identically.

PPM. The process performance manager (PPM) supports the management of business processes and was developed by the IDS Sheer AG (2000). The PPM tool was developed in order to (1) guarantee compliance with activities and effort plans, (2) identify weak points, (3) optimize the business process by identifying improvement potential, and (4) assess the achieved improvements on the business process. Therefore, it provides a basis for decision making within an organization.

The PPM focuses on closing the feedback gap between business process specification and execution. It provides functions for observing and assessing the performance of a current business process, and for providing feedback about it. Furthermore, the PPM is able to integrate existing (organization-specific) tools, and therefore, is able to present a common view across heterogeneous systems. Key performance indicators (KPIs) characterize a business process across different aggregation hierarchies and can be viewed according to different filters. Through baseline specification, statistical analysis, and trend identification, the PPM is able to identify deviations from baselines and to inform decision-makers.

However, there are no concepts for providing a role-oriented presentation and visualization of measured data nor for enhancing the set of provided functions.

PAMPA. The PAMPA (project attribute monitoring and prediction associate) tool (Simmons et al., 1998) is especially designed for data collection and visualization. It supports the work of a project manager by enhancing intellectual control over the software development project.

PAMPA is integrated into a dual control and improvement cycle. It implements the project visualization stage, which consists of (1) data collection and (2) data analysis and prediction. This stage could easily be integrated into the control and improvement cycle of a certain project. Intelligent agents reduce the overhead of data collection significantly. They replace manual and subjective data collection and analysis by objective procedures and allow a cost effective, automated solution for project control. Agents are responsible for data collection and analysis, for the
generation of reports, and they inform project personnel in case of plan deviations. Agents are generated by expert systems, which get their inputs from the PAMPA system.

PAMPA uses intelligent agents to reduce the overhead of data collection and analysis. It provides a set of predefined objects with relationships and attributes, which are instantiated in the context of each project. Presentation and visualization depend upon Microsoft Windows and Office.

4.3 Relation to the Operational SPCC Concept

Section 3 introduced the operational SPCC concept and gave some ideas on what a logical SPCC architecture could look like and on how to implement an SPCC. All of the previously discussed approaches could be seen as a kind of SPCC instantiation; that is, they fill the SPCC concept with life in very different ways and cover several dimensions of the identified taxonomy of Section 2 differently. For instance, if we have a look at the purpose dimension, one approach could focus on monitoring and comparison while another approach could focus on analysis and guidance, or one approach could use different analysis techniques and methods than another approach.

Each of the approaches discussed has a different underlying architecture and different ways of performing the control over a software development project, but they all use certain techniques or methods (so-called SPCC functions in Section 2.1) in order to implement various purposes such as monitoring, comparison, analysis, and guidance. That is, we are able to identify the techniques and methods used for each particular approach. In this section we want to sketch four selected techniques and methods that can basically be used for analysis and guidance.

Classification Tree Analysis. Classification tree analysis is a widely-used statistical method that is used in the context of Amadeus to identify error-prone or high-risk software components on the basis of previous software releases (Porter and Selby, 1990). In order to build up a classification tree one has to define so-called target classes (e.g., all components with more than n interface errors). After that, a recursive algorithm searches metrics for each node of the tree to distinguish between components inside and outside the target class based upon components of previous software releases. That is, the root of the classification tree contains all components to be classified, the nodes contain partly classified components, and finally, the leaves contain components that are either inside or outside the target class.

Dynamic Variables. Doerflinger and Basili (1983) describe the use of dynamic variables as a tool for monitoring software development. The idea is to assume underlying relationships that are invariant between similar projects. These relationships are used to predict the behavior of projects with similar characteristics. A baseline of an observed variable is generated from measurement data of one or more completed projects in order to compare projects in progress against it. For instance, some projects with a representative productivity might be grouped to form a productivity baseline. 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., the predetermined tolerable variation from the baseline), the project manager is alerted and has to determine the possible reasons for the failure. The method to determine most probable deviation causes is as follows: (1) Flag
any measure outside an appropriate tolerance range. (2) Analyze the appropriate parts of an associated table with possible interpretations of the deviation for each flagged measure. (3) Count overlaps of possible interpretations; that is, count the number of emergence of a certain interpretation in every flagged table. (4) Determine the most probable interpretation; that is, the interpretation with most overlaps. If one interpretation appears more often than another, the former is more probable than the latter.

**Cluster Analysis.** Li and Zelkowitz (1993) describe the use of cluster analysis for extracting baselines from collected software development data. This data is collected manually, such as effort data, error data, and subjective and objective facts about projects, or automatically, such as computer use, program static analysis, and source line counts. Each measure of a project is described by a so-called measure pattern, which is represented by a 15-dimensional vector composed of measurement values for 15 points of time. Cluster analysis is a technique for finding groups in data that represent the same behavior. It is used to find similar measure patterns within collected data. The group of similar patterns is called a cluster. A cluster model is the average of all measure patterns within one cluster. For instance, you have a set of \( n \) measure patterns, represented by 15-dimensional vectors \( (p_{1,1}, \ldots, p_{1,15}) \) to \( (p_{n,1}, \ldots, p_{n,15}) \). The model of this set is computed by:

\[
(m_1, \ldots, m_{15}), \text{ where } m_i = \frac{p_{1,i} + \cdots + p_{n,i}}{n} \quad \text{for } i \in \{1, \ldots, 15\}.
\]

A cluster consists of at least three measure patterns. Two patterns belong to the same cluster if their Euclidian distance is less than a certain threshold. For a project in progress, a manager estimates the values of some variable to build an initial measure pattern. Bit by bit, estimates are replaced by real data as soon as they become available, and so the cluster model that is closest to the measure may change continuously. Furthermore, general characteristics for a new project within a given cluster can be determined from the common characteristics of old projects within the same cluster.

**Identification of Trend Changes.** If a multitude of data points is available, the problem of making a decision based on collected project data is very difficult. The resulting scatter plot is hard to analyze and trend changes can not be identified. Tesoriero and Zelkowitz (1998b) describe a method of smoothing data and identifying relevant trend changes. The basic idea is to compute the so-called exponential moving average (EMA) to smooth a given scatter plot. The method is adapted from the financial community and is used on sample data from NASA/SEL. Basically, the technique consists of three steps. (1) At first, a smoothing technique is used to approximate the behavior of the data. Therefore, we use the EMA algorithm to smooth the scatter plot. (2) After smoothing the scatter plot, we need to compute the extreme values of the resulting curve in order to identify trend changes of the original scatter plot. The smoothed scatter plot is continuous, but it is not derivable. Therefore, another efficient way to compute the extreme values (called pivot points) is chosen. (This step is based on instantaneous derivatives and is a second invocation of the EMA algorithm itself.) (3) The last step is to connect each segment, defined by the pivot points, by a straight line. The resulting linear function is called characteristic curve and represents the major trends and trend changes of the original scatter plot.
5 Summary and Future Research Directions

This article conceptually integrates an SPCC into the context of learning and improvement-oriented organizations. For doing so, we used the TAME model, which integrates measurement, support for model building, and support for project control and guidance. An SPCC is deployed during the execution of a software development project. Underlying essential principles are measurement-based control, purpose- and role-oriented interpretation and visualization, and the use of experience.

Based on these assumptions a description of main definitions for SPCCs is given in order to provide a consistent framework. The description distinguishes five dimensions according to purpose, technical, improvement, role, and tool aspects.

Based on this terminology the operational concept of an SPCC is discussed, the underlying software development model is mentioned, a two-dimensional characterization schema for SPCC is introduced, and a possible logical architecture is proposed. The proclaimed architecture supports (1) a variable set of techniques and methods to provide the basic functionality, (2) context-oriented, project-specific adaptations of functions, and (3) a goal-oriented presentation according to a certain usage goal of the SPCC. Furthermore, integration of popular visualization tools is possible.

Selected existing approaches for SPCCs in the context of the described framework are sketched and classified along the characterization schema. Existing approaches offer only partial solutions for SPCCs; that is, they can be seen as a partial instantiation of our SPCC concept. Especially purpose- and role-oriented usages based on a flexible set of techniques and methods are not comprehensively supported. This leads to future research directions:

One important research issue is the development of a schema for adaptable SPCC functions, which effectively allows for purpose-driven usage of an SPCC in varying application contexts. Another research issue is the elicitation of information needs for involved roles and the development of mechanisms for generating adequate role-oriented visualizations of the project data.

Open questions refer to the combination of different techniques and methods for process-accompanying data processing and interpretation. If a variable set of techniques and methods exist, guidelines are needed on how to combine techniques according to a certain SPCC usage goal. Furthermore, a method is needed to choose appropriate techniques with respect to a certain purpose, combine and adapt the chosen techniques to the project characteristics, and present and visualize the resulting data according to a certain viewpoint. Moreover, unified representation and specification schema for techniques and methods would facilitate the development of a consistent and expandable set of functions.

Another important research issue is the support of an SPCC for change management. When the goals or characteristics of a project change, the real processes react accordingly. Consequently, the control mechanisms, which should always reflect the real world situation, must be updated. This requires flexible mechanisms that allow for reacting on process variations. General problems exist in keeping data collection procedures consistent with the real processes, and managing partial backtracking of process tracks in the case of erroneous process performance. One open question is how to control process backtracking or on-line refinement during project execution, which is especially important for controlling “creative” process elements.
One long-term goal of engineering-style software development is to control and forecast the impact of process changes and adjustments on the quality of produced software artefacts and other important project goals. An SPCC can be seen as a valuable contribution towards reaching this goal. Furthermore, the authors expect, that the described concepts for SPCCs can be adapted for other domains (i.e., business processes) and might enforce the establishment of engineering-style and reuse-oriented project control centers (PCCs).

Acknowledgements

We would like to thank Prof. Dr. Dieter Rombach from the Fraunhofer Institute for Experimental Software Engineering (IESE) for his valuable comments to this article, and Sonnhild Namingha from the Fraunhofer Institute for Experimental Software Engineering (IESE) and Marcus Ciolkowski from the University of Kaiserslautern for reviewing the first version of the article. Furthermore, we would like to thank the anonymous reviewers for their helpful comments on this article.

References


Briand, L.C., Differding, C., Rombach, H.D., 1996. Practical Guidelines for Measurement-


Lott, C.M., 1993. Process and measurement support in SEEs. ACM Software Engineering
5 SUMMARY AND FUTURE RESEARCH DIRECTIONS

Notes 18(4), 83-93.


Biography

Jürgen Münch. Jürgen Münch received the B.S. degree (Vordiplom) and the M.S. degree (Diplom) in computer science with a minor in economics from the University of Kaiserslautern, Germany. He is executive board member of the temporary research institute SFB 501 “Development of Large Systems with Generic Methods” funded by the German Research Foundation (DFG). In January 2001, he joined the Fraunhofer Institute for Experimental Software Engineering as a competence manager. Before that he was a researcher at the Software Engineering Research Group (AGSE) at the University of Kaiserslautern, Germany. His research interests are in software methodologies, modeling and measurement of the software process and resulting products, and reuse of software process models. Jürgen Münch is a affiliate of the IEEE Computer Society and member of the German Computer Society (GI).

Jens Heidrich. Jens Heidrich received the B.S. degree (Vordiplom) and the M.S. degree (Diplom) in computer science with a minor in mathematics from the University of Kaiserslautern, Germany. He is member of the temporary research institute SFB 501 “Development of Large Systems with Generic Methods” funded by the German Research Foundation (DFG). Since July 2001, he is a researcher at the Software Engineering Research Group (AGSE) at the University of Kaiserslautern, Germany. His current research interests include software project management, software development methodologies, quantitative analysis and evaluation of software development processes and products.