Reference:
Jürgen Münch, Jens Heidrich. Handbook of Software Engineering and Knowledge Engineering, volume 3, chapter Tool-based Software Project Control, pages 477-512. World Scientific Publishing Company, 2005.
DOI: 10.1142/9789812775245_0017
URL: http://www.worldscientific.com/doi/abs/10.1142/9789812775245_0017

Excerpt from "Handbook of Software Engineering & Knowledge Engineering, Vol. 3 Recent Advances", ISBN 981-256-273-7, World Scientific 2005, pp. 477-512

TOOL-BASED SOFTWARE PROJECT CONTROL

JÜRGEN MÜNCH* and JENS HEIDRICH†

Fraunhofer Institute for Experimental Software Engineering,
Sauerwiesen 6, 67661 Kaiserslautern, Germany
*E-mails: *muench@iese.fraunhofer.de*
†heidrich@iese.fraunhofer.de

Developing software and systems in a way that entails plannable project execution and predictable product quality requires the use of quantitative data for project control. In the context of software development, few techniques exist for supporting online monitoring, interpretation, and visualization of project data. This is caused particularly by the often insufficient use of such engineering principles as experience-based planning and plan-based execution in the software development domain. However, effective software project control requires integrated tool support for capturing, managing, analyzing, and storing data. In addition, advanced control approaches aim at providing purpose- and role-oriented information to all involved parties (e.g., project manager, quality assurer) during the execution of a project. This chapter introduces the concept of a so-called Software Project Control Center (SPCC), sketches a control-oriented software development model, and gives a representative overview of existing tool-based software control approaches from academia and practice. Finally, the different approaches are classified and compared with respect to a characterization schema that reflects important requirements from the viewpoint of practitioners.

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

1 Introduction

The rapidly increasing importance of software in today's business and the high dependability on software in contexts such as critical systems demands answers on how software can be developed in a plannable way (i.e., matching time and budget constraints), whereby the resulting software product matches predefined functional and non-functional requirements (i.e., reliability). Outsourcing and global distribution of software development activities as well as the integrated development of systems with hardware and software components make predictable software project execution even more necessary. In addition, having repeatable processes for developing software variants within accurate time and cost estimations is gaining increasing importance. Modern cars, for instance, contain more than 50 software-based controllers. Missed delivery deadlines for such controller software can delay system integration significantly and lengthen the overall development; providing inadequate software quality can lead to expensive callbacks of thousands of vehicles. Many software development organizations still lack support for obtaining intellectual control over their software development projects and for determining

* Corresponding author.

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.

Software development based on engineering principles [13] [34] addresses these problems. It includes the definition of project goals, the development of explicit project plans based on experience [32], the execution of projects based on these plans, and the packaging of experience for future projects. The packaging step also includes the creation of explicit models (e.g., an effort baseline) from past project data in order to be reused and adapted while planning a new project. This feedback-cycle involves Software Engineering as well as Knowledge Engineering aspects.

One important means of engineering-style development is software project control. This comprises monitoring and analysis of actual product and process states, comparisons with planned values and the initiation of corrective actions during project execution. Measurement technology is needed to derive metrics for process and product characteristics, define target data values, and establish models for data comparisons and predictions.

In the mechanical production domain, so-called control rooms are a well-known instrument for engineers. A control room is a central node for all incoming information of a production process. It collects all incoming data (e.g., the current state of an assembly line) and visualizes them for control 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. Controlling in the software engineering domain requires an analogue approach that is tailored to the specifics of software processes (such as non-deterministic, concurrent, and distributed processes). A control room in the software engineering domain is a so-called Software Project Control Center (SPCC) [28]. Typical tasks of SPCCs such as distributed data collection, data storage, and data analysis suggest the use of tool-support for performing the tasks. Nowadays, systematic software project control is finding its way into practice. This is very often accelerated by organizational efforts to reach higher maturity of their software processes and practices.

It should be mentioned that software project control can not only be used to monitor and adjust project performance during execution. The collected data can also be packaged (e.g., as predictive models) for future use and contribute to an improvement cycle spanning a series of projects. Additionally, data collection can be extended beyond the purpose of controlling. One example would be to perform a case study as part of a development project in order to determine the effects of process improvement.

This chapter is organized as follows: The next section gives a basic overview of software project control by means of a development model that integrates SPCCs into the software development cycle. Section 3 describes the scope of the discussed tool-based SPCC approaches. Section 4 introduces basic requirements

for SPCCs. Section 5 discusses a set of tool-based SPCC approaches referring to the requirements. Section 6 illustrates some integrated SPCC techniques. Finally, Section 7 classifies the discussed approaches and highlights future research issues in this field.

2 Software Project Control

The activities around a software development project can be grouped into three basic phases. First, we consider activities related to project planning, including the basic characterization of the project itself, goal setting, such as setting measurement goals according to the Goal Question Metric (GQM) paradigm [1] [2] [5] [8] [10] [31] [36], and finally, selection of the right development process. Second, we consider activities related to project execution, including all development activities as well as project and quality management activities. Third, we consider activities related to know-how management, including activities for the analysis of project data, and packaging data in order to reuse them in future projects.[†]

Generally, project control can be defined as ensuring that project objectives are met by monitoring and measuring progress regularly to identify variances from plan so that corrective action can be taken when necessary [29]. Controlling is an activity that is basically applied during project execution, but has strong ties to planning and know-how management. Planning is the basis for controlling in order to be able to identify variances from plan. Know-how management is built upon controlling in order to be able to analyze the monitored and measured progress of the project and to package these data for future projects.

Sometimes controlling is defined as the pure monitoring and measurement process during project execution, and all corrective actions to bring the project back to plan, the so-called steering activities, are excluded. For this chapter, we want to include all such steering activities explicitly in our definition of the term "project control".

A Software Project Control Center (SPCC) supports project control and is defined as a means for process-accompanying interpretation and visualization of measurement data. An SPCC consists of a control architecture that clearly defines interfaces to its environment, and a set of underlying techniques and methods that allow for project control. Basically, an SPCC retrieves input data from the current project (e.g., goals, characteristics, baselines, and current measurement data) and generalized data from previous projects (e.g., quality, product, and process models), and produces a visualization of measurement data by using the incorporated techniques and methods to interpret the data accordingly. An SPCC is itself a kind of general control approach according to our definition and not necessarily tool-supported. But in order to fulfill control tasks like monitoring defect profiles, de-

[†] All activities mentioned during the three phases cover the six project-related steps of the Quality Improvement Paradigm (QIP) [7].

tecting abnormal effort deviations, cost estimation, and cause analysis of plan deviations, a certain amount of tool support is necessary and inevitable. Therefore, this chapter focuses on tool-supported software project control and highlights some approaches in this field.

SPCCs can be classified along two dimensions. First, the degree of variability and adaptability of used techniques and methods to control the project, which can vary from a strictly predefined set of built-in techniques and methods to a complete extensible and adaptable repository of techniques and methods. Second, the goal orientation of the produced views onto monitored measurement data, which can vary from one static view for each monitored project variable (like one chart for the project's effort) to a set of goal-oriented views, that is, views that depend on the previously defined measurement goals (like a set of charts covering all previously defined measurement goals).

The TAME (Tailoring a measurement environment) software development model [6] [7] acts as a basis for integrating an SPCC into a Software Engineering Environment (SEE). It makes all necessary control 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 (GQM) method and the Quality Improvement Paradigm (QIP) as well as the Experience Factory (EF) [3] [4] [12] organization. The model has been instantiated in several progressive software engineering environments, such as NASA-SEL [24] or MVP-E [9].

Fig. 1 shows an integrated development model for SEEs based on the principles of the TAME model. It is used here to illustrate how an SPCC could be integrated into a software development organization and how it interacts with its environment. The development model distinguishes four different levels of abstraction: (1) roles, (2) services, (3) tools, and (4) information. The presented model exemplarily assigns several units to each of the four levels in order to illustrate the integration of an SPCC within a software development project. According to [28], we can describe the four levels as follows:

Roles: Different roles within a software development project use different services of the three basic phases of the project. For instance, the project planning group uses planning services in order to create a project plan, the development group uses technical services in order to develop products, the project management uses management services in order to control the project, and so on.

Services: This level incorporates services that support activities of the three basic project phases. (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. We distinguish initial planning, which refers to planning before the start of the project, from replanning, which addresses the systematic changing or detailing of the plan during project execution

(eventually supported by an SPCC). (2) The services mainly needed for project execution can be divided into services for technical development of products, services for project management (including project control), and, finally, services for quality assurance. (3) Know-how management is used to analyze the collected project data in order to provide and improve existing models for future use.

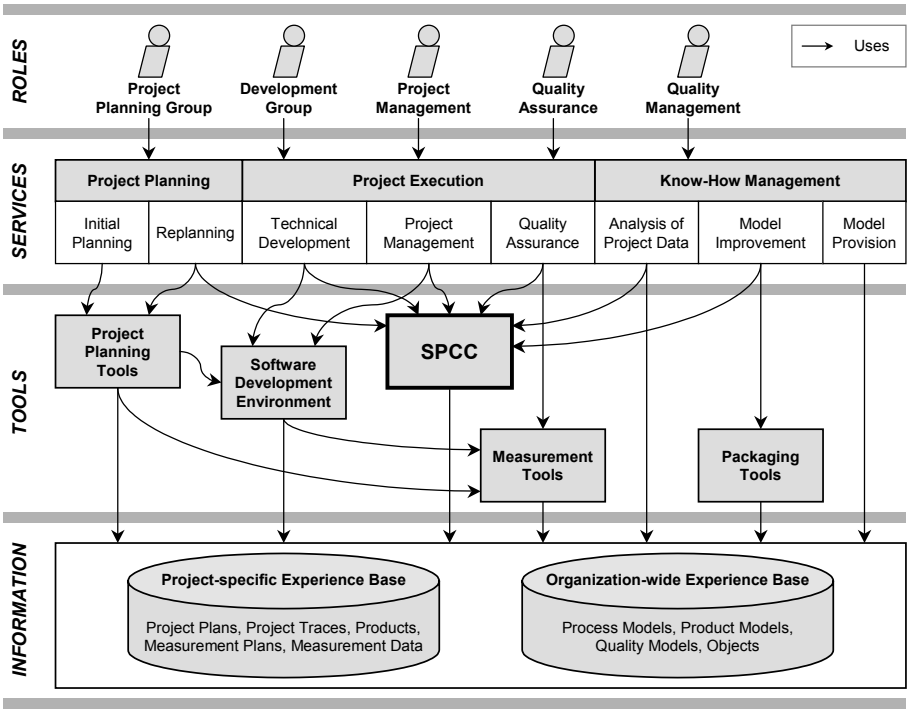

Fig. 1. Software development model.

Tools: A set of tools is able to provide some services completely (fully automated) or to support certain aspects of a service (human-based tool application). For instance, there exist tools to support project planning, to measure project data, to develop artifacts, or to package experience. The tools are invoked by services and use information resulting from other tools or from an experience base to perform their tasks. An SPCC tool supports roles such as project management, quality assurance, or development group (respectively corresponding services) by providing, processing, interpreting, and visualizing process-accompanying data for their specific needs and purposes. Thereby, it builds upon information stored in an Experience Base located in the last level of our development model.

Information: The fourth level 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 informa-

tion level is to provide project-specific or organization-wide information during project execution in order to improve control over the project.

3 Scope of the Overview

This chapter presents an overview of selected tool-based approaches in the SPCC field. Existing surveys reside on a specific level of abstraction and present the work from a specific perspective. There are papers describing frameworks (e.g., [22]), papers describing methods and techniques (e.g., [38]), and papers describing tools (e.g., [17]). 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).

	Product-oriented Controlling	Process-oriented Controlling	Resource-oriented Controlling
Software Development Domain	X	X	
Business Domain	(X)	(X)	
Production Domain			

Fig. 2. Scope of the overview.

In this chapter, we define the scope in the following way: We focus on tool-supported approaches for online data interpretation and visualization in the context of software development (see Fig. 2). We consider the selected approaches for the overview as being 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 control 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), there exist a lot of control approaches (e.g., performance management with user-configured dashboards, which give the user the ability to view required information in a user-specific way). 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 of its use in the software development domain.

Second, we exclude solely resource-oriented approaches (such as attendance/absenteeism control, person-job matching, and 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), function plotting programs (e.g., gnuplot), and purely data monitoring-oriented approaches (e.g., the design and code metrics included in Borland Together). 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. Purely data monitoring-oriented approaches can be used as one way to gain data for further SPCC interpretation and visualization activities.

This overview touches different abstraction levels because narrowing to only one level (e.g., only focusing on the tool level) would contradict our goal of providing a comprehensive overview of essential approaches for software project control centers. The description of the approaches focuses on the logical architecture to highlight the main components and their interaction. Additionally, selected techniques and methods in the context of the particular approach are sketched.

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 control tools.

In the following, we will concisely describe the intended use, the main idea, the logical architecture, and associated techniques, methods, and tools of each approach, where appropriate.

4 SPCC Requirements

In order to discuss the related tool-based approaches according to a unique schema, we will introduce some basic requirements for SPCCs. The description is organized as follows: At first, we will discuss requirements in the field of data collection; that is, requirements referring to distributed development, database access, and so on. Afterwards, we will treat data processing requirements; that is, requirements referring to the external and internal SPCC data processing functionality. Then, we will discuss requirements referring to data presentation, like abstraction and compression of data and viewpoint-oriented presentation.

4.1 Data Collection Requirements

The following section gives an overview of requirements in the context of the overall SPCC data collection process; that is, it captures all requirements for incoming data.

R1: Support for reuse. All data of previous projects (like effort and error distributions) and organization wide experience (like quality models and qualitative experience) are potential reuse candidates. Project-specific data has to be adapted according to project goals and characteristics and models have to be instantiated in the context of the current project. This leads to a generic reuse architecture in which all used data are customized in correspondence to current project goals and characteristics. An SPCC should support such a generic reuse architecture.

R2: Support for distributed development. Software is increasingly developed at different distributed development locations. Each environment has its own way to collect data and to store them in a corresponding repository. An SPCC has to support distributed development and has to be able to integrate the data from different development environments into one single framework.

R3: Integration of measurement paradigms. There are several approaches to derive measurement plans for a project. One goal-oriented top-down approach is the already mentioned GQM method, which systematically derives questions from measurement goals and metrics from derived questions. An SPCC has to take account of measurement goals and plans and use them as an input for setting up the required SPCC functionality. So, an SPCC has to provide concepts in order to integrate an underlying measurement paradigm.

R4: Data validation capabilities. Incoming data have to be validated with respect to whether they make sense in a special (project) context or not. This includes for instance the range of incoming data. Furthermore, data from different distributed development locations have to be consistent in order to be integrated for further data processing. In general, an SPCC has to provide mechanisms for testing the validity of incoming data.

4.2 Data Processing Requirements

The following section gives an overview of requirements in the context of the SPCC data processing functionality.

R5: Provision of control techniques. We distinguish different usage purposes of an SPCC. For instance, an SPCC can be used to compare data with a corresponding baseline and detect deviations. For each purpose (or a group of purposes) a variety of possible techniques and methods exists. An SPCC has to provide a comprehensive pool of predefined techniques in order to control the project.

R6: Packaging of experience. In order to support organization-wide improvement paradigms, we need to integrate experiences, gained through the usage of an SPCC, into an experience base. This includes quantitative experience (e.g., an effort distribution) as well as qualitative experience (e.g., a list of retaliatory actions which has proofed to be useful if certain plan deviations occur). Therefore an SPCC has to provide concepts in order to package experiences for further reuse.

R7: Variability. The applicability of a certain control technique often depends on the project's context and the attributes (e.g., effort, number of defects, or design complexity) to control in this context. For instance, in order to apply Statistical Process Control the values of a controlled process attribute have to vary around an expected value. So, we'll probably need new methods and techniques if we introduce new project types, or new attributes to control. Therefore, an SPCC has to provide an extensible pool of techniques; that is, it has to be able to integrate new techniques.

R8: Adaptability. Every project differs in terms of project goals and characteristics (e.g., domain, programming language, or skills of developers). Therefore, all control techniques have to be adapted accordingly and an SPCC has to provide corresponding mechanisms to do so. Furthermore, it is possible, that certain project characteristics change during project execution. In this case the applied control techniques have to be adapted to the new project characteristics dynamically.

4.3 Data Visualization Requirements

The following section gives an overview of requirements in the context of the SPCC data visualization process.

R9: Goal-oriented visualization. Different users of an SPCC (like a project manager, a quality assurance manager, a developer, or a tester) need different kinds of data visualizations. For instance, a project manager will probably need all data related to the state of the project, while a module designer will probably only need data about the design quality of a certain module. Therefore, an SPCC has to provide concepts for goal-oriented data visualization.

R10: Support for data compression/abstraction. SPCC users need data on different levels of abstraction at different points in time. For instance, a project manager needs an overview of the project's effort per project phase. If a plan deviation

occurs in a certain phase, it is necessary to have a closer look at the effort data for all activities of this phase, and so on. A module designer, for instance, just needs design quality values for a certain module, while the system designer needs them for the whole software system. Generally, an SPCC has to provide mechanisms in order to present the processed data on different levels of abstraction.

R11: Provision of up-to-date information. In order to interpret the visualized data in the right way, you have to be sure, that all displayed data are up-to-date. An SPCC has to guarantee this. This requires traceability among (1) the collected measurement data, (2) the way they are processed (by control techniques), and finally (3) the way they are visualized (for different users). Then, it is possible to trace the overall data flow, which allows for interpretation of visualized data in the right way (e.g., in order to find causes for plan deviations).

5 Tool-based Software Project Control Approaches

In the following we discuss a set of tool-based approaches for software project control. First, we'll briefly address the idea behind each approach and the field of application. Second, we'll describe the architecture used to implement the approach. Third, we'll discuss the requirements from Section 4. An overview of the latter is presented in Table 1. The discussion of requirements for each approach is organized into three parts. Part (a) mentions requirements fully supported by the approach, indicated by "Fully" in Table 1. Part (b) discusses requirements only partly supported; that is, the basic concepts are provided, but the integration is not fully addressed, indicated by "Partly" in Table 1. If necessary, part (c) lists requirements not addressed by the corresponding approach, indicated by an empty cell in Table 1.

5.1 Provence

Idea: Provence [20] is a framework for project management. It informs managers about state changes of processes and products, and is able to generate reports of the project. Furthermore, it allows project managers to initiate dynamic replanning steps. The main idea behind Provence 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 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.

Architecture: The logical architecture of Provence consists of five components and is shown in Fig. 3. (1) A process server supports process transitions according to a specific process model. It generates specifications of requested events and registers those events at the event action engine. (2) The data management unit

stores all relevant data of processes and products and allows the user to query the corresponding database. It is based on a pre-defined data model. (3) A smart file system detects changes of files caused, for instance, by tool invocation, and submits these events to the event action engine. (4) The event action engine collects all incoming events and informs the process server in case of previously registered events. (5) Finally, the visualizer is responsible for visualizing current process and product data. A prototypical instantiation of the Provence architecture uses existing tools to implement the five logical components of Fig. 3 (e.g., the process-centered SDE Marvel [19] is used as the process server).

Table 1. Overview of all discussed approaches.

Approach	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11
Provence		Partly		Partly	Fully					Partly	Fully
Amadeus	Partly	Fully	Partly	Partly	Fully	Partly	Fully	Fully	Partly	Partly	Fully
Ginger2			Partly	Partly	Fully	Fully	Partly	Partly	Partly		Partly
SME	Fully		Partly	Partly	Fully	Fully		Partly	Fully		Fully
WebME	Fully	Fully	Partly	Fully	Fully		Partly	Fully	Partly	Fully	Fully
PPM	Partly	Partly		Partly	Fully	Partly	Partly	Fully	Partly		Fully
PAMPA		Fully	Partly		Fully	Partly	Partly	Partly			Fully
SPCC Tech.	Partly	Fully	Partly	Fully	Fully	Partly	Fully	Fully	Fully	Fully	Fully

Requirements: (a) Provence provides a set of standard control techniques in order to check for process conformance. The component-based architecture allows integration of alternative tools for each component. The visualizer guarantees an up-to-date view of process and project data. (b) Depending on the used tool-set it should be possible to support distributed software development and data validation mechanisms. Depending on the used visualizer data could be displayed on different levels of abstraction. (c) However, Provence provides no access capabilities to former project data or experience gained from former projects. It further provides no support to integrate existing measurement paradigms. Current project information cannot be packaged in order to be reused in future projects, and a static set of integrated standard queries for visualizing the project state is provided. Moreover, no mechanisms to support goal-oriented data visualization for different project roles and adaptation of applied analysis techniques are addressed.

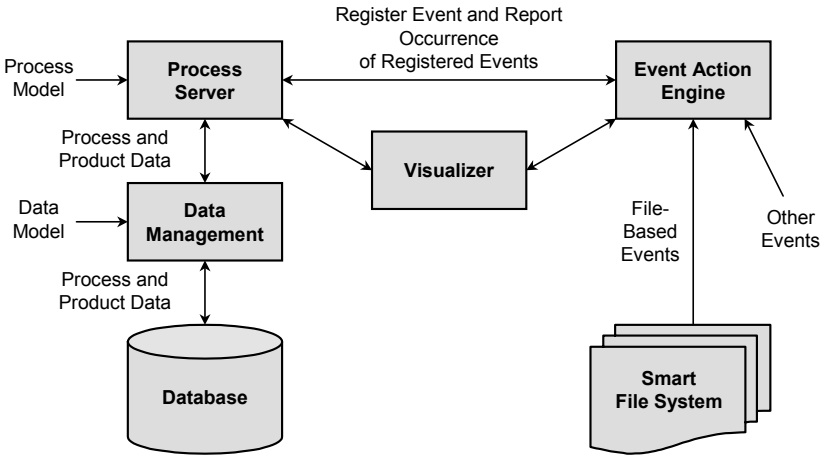

Fig. 3. Logical architecture of Provence [20].

5.2 Amadeus

Idea: Amadeus [30] [33] is a metric-based analysis and feedback system, and 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 for interpreting different types of events. The main idea of the Amadeus system is to make measurement an active component during project execution. Amadeus is based on a script language that dynamically interprets process events, object state changes, and calendar time abstractions. These three events can be combined to form 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 either by humans or automatically. For instance, a method called classification tree analysis is used to classify components of a software system.

Architecture: The main part of the logical architecture (shown in Fig. 4) is a pro-active server, which interprets scripts and coordinates event observation and agent activation. Data integration frameworks allow for collecting data from processes, products, or personnel. The server does not distinguish between scripts generated by users, processes, or tools. A number of servers is able to run and interact (via scripts) at the same time. A server interacts with the evaluation component of an agent and coordinates tasks, where different agents participate. A client communicates with a server via dialog boxes. Both server and client have an associated expandable tool kit. The client's tool kit includes tools to define a script, and the server's tool kit includes tools as part of script interpretation (i.e., data collection and analysis tools). The user interacts with a so-called customizable goal palette that provides a summary of all available services and analysis processes. After selecting one process the system guides the user through the chosen technique.

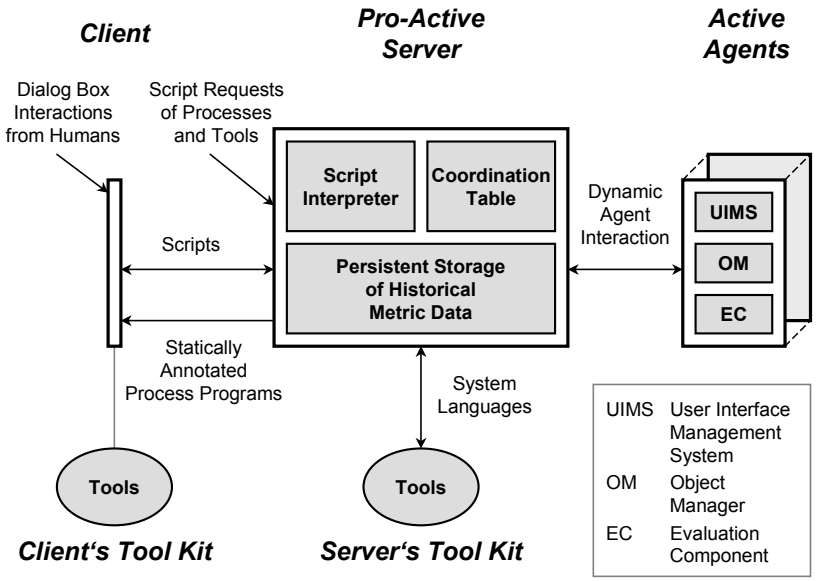

Fig. 4. Logical architecture of Amadeus [33].

Requirements: (a) The flexible server/client architecture allows Amadeus to support distributed software development. Client and server tool kits provide a set of standard techniques for project control and allow easy integration of new techniques and adaptation of existing ones for different projects. The user's goal palettes allow an up-to-date view on gathered measurement data and analysis results. (b) Depending on the used techniques as part of the server and client tool kits accessing a reuse repository and packaging project experience should be possible. The same holds for data validation capabilities and integration of measurement paradigms. Amadeus users are individually supported by customizable goal palettes including available services and analysis processes. However, no user-dependent views of analysis results are addressed.

5.3 Ginger2

Idea: Ginger2 [40] [41] 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, consisting of questions and hypotheses. The goal of the CAESE approach is to find knowledge about the given problem statement. Fig. 5 illustrates the principal design of a CAESE environment. In that, 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).

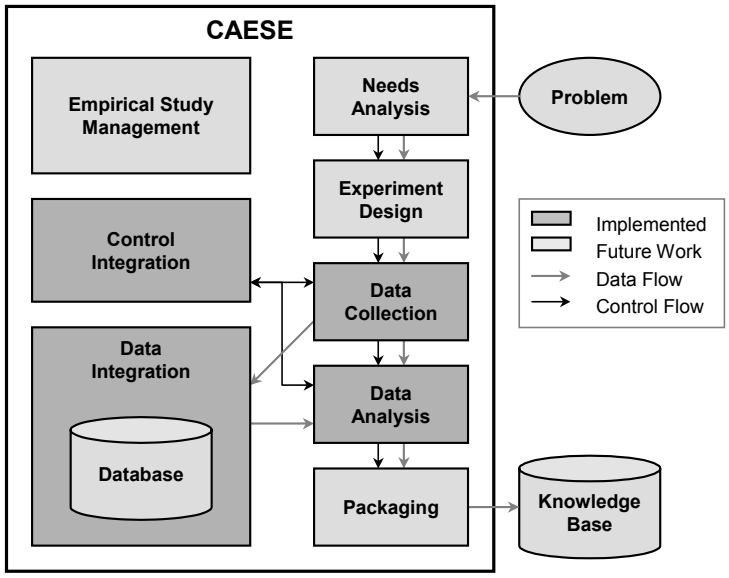

Fig. 5. Logical architecture of Ginger2 [41].

Architecture: The life cycle model of a CAESE environment consists of the following parts: The *needs analysis* unit is used to identify the problem, formulate goals and the purpose of the study, and establish hypotheses. The *experiment design* unit determines how to enact the experiment (i.e., which participants are selected and which techniques, methods, and tools are used). The *data collection* unit gathers data according to the experiment design and a data collection model, which is used to collect data from different views. The *data analysis* unit analyzes the data according to the goals and the purpose of the study. Finally, the *packaging* unit packages the problem statement and the analysis results into a knowledge base. In addition, there are some elements that support the units of the life cycle model: A *data integration* unit transforms data from the data collection unit according to the input type of the data analysis unit (e.g., making continuous data discrete). The *control integration* unit synchronizes the usage of different tools. Finally, the whole study is managed by the *empirical study management*. Ginger2 uses a multitude of techniques to collect data. Most of them are used to observe the behavior of the experiment participants. For instance, there are techniques to gather audio and video data of the experiment, data about mouse movements, keyboard inputs, and window movements, data about eye tracking, motion, and skin resistance of the participants, and finally, data about tool usage and program changes[‡].

Requirements: (a) The Ginger2 environment provides a set of standard techniques for gathering and analyzing measurement data and allows for packaging

[‡]The collection of data about tool usage and program changes is the main focus of the Ginger1 system [40].

gathered knowledge in a knowledge base. The needs analysis unit allows for specifying goals and purposes of a study. (b) However, there is only limited integration of a measurement paradigm as well as mechanisms for data validation and goal-oriented visualization depending on the data collection and analysis units. The same holds for variability and adaptability of integrated techniques and methods. The focus of the Ginger2 system is on learning by means of controlled experiments. As a matter of facts, lots of analysis results are only available and used after project completion. (c) The lifecycle model does not address support for reuse and support for distributed development of software.

5.4 SME

Idea: The software management environment (SME) [16] [17] was developed within the software engineering laboratory (SEL) [21] [24] 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. Its basic functions are observation, comparison, prediction, analysis, assessment, planning, and control.

Architecture: The implementation of SME functions relies on information from previous projects, research results from studies of software development projects, and management rules, which are accessible via three separate databases. The architecture is presented in Fig. 6. 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. 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. All these data are input for the SME, which performs the management functions above. These functions provide data for the project manager in order to support well-founded decision making. Experience gained during project execution may lead to changes of project data. This feedback mechanism enables SME to work with up-to-date information.

Requirements: (a) The SME provides access to different kinds of reuse repositories and supports packaging of project information. It further provides a set of standard management techniques covering different purposes (e.g., guidance). Integrated feedback mechanisms guarantee up-to-date project information. The goal-oriented visualization of gathered project data is mainly focused on the project manager. (b) SME supports a static set of collected measurement data corre-

sponding to previously defined (static) measurement plan. However, mechanisms exist to adapt the SME to the needs of the currently performed project and to validate ingoing data. (c) Distributed development, different levels of abstraction for data visualization, as well as the enhancement of integrated techniques is not addressed.

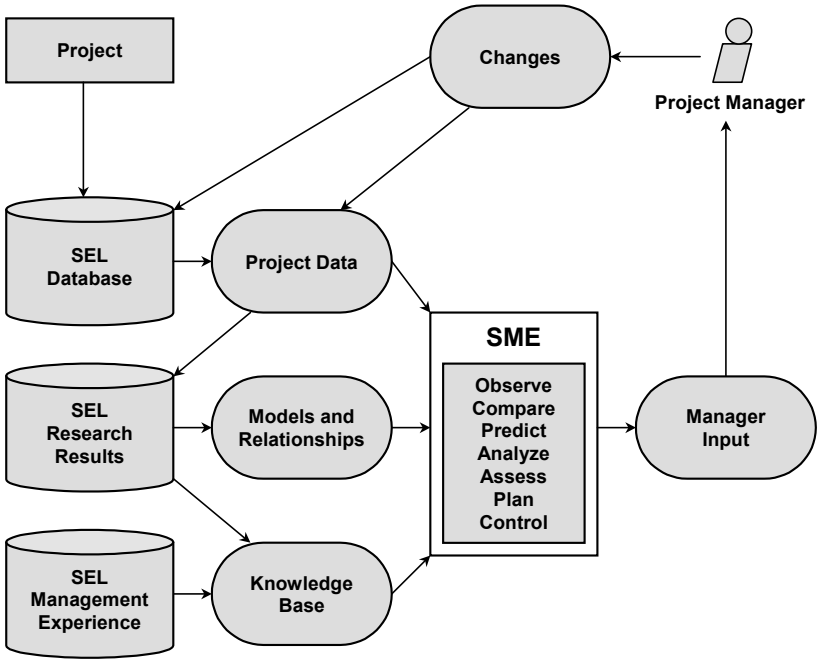

Fig. 6. Logical architecture of the SME [16].

5.5 WebME

Idea: WebME (web measurement environment) [37] [38] [39] is a web-based data visualization tool that is based on the SME approach. WebME enhances the capabilities of SME in terms of distributed development of software. While SME is concentrated on the development within a certain closed SDE, WebME supports the development at different locations with heterogeneous SDEs and provides appropriate data integration mechanisms. WebME uses a special script language, which is able to integrate information of different heterogeneous SDEs into one common view.

Architecture: The architecture is presented in Fig. 7. It consists of three layers. The *end-user applications* layer provides access to the WebME system via a web browser. A user specifies a query and gets an HTML response. 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. The query processor receives data from the web browser and trans-

forms 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. 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.

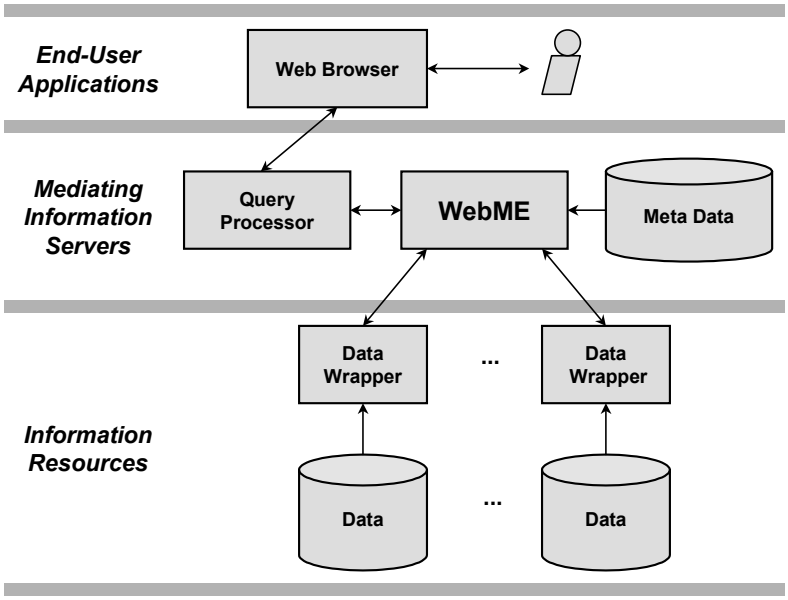

Fig. 7. Logical architecture of WebME [38].

WebME uses the so-called *data definition language (DDL)* to specify scripts that describe which data of which host and in which format are to be used and how this data can be presented according to a certain query. A script processor transforms the scripts into measurement class and interface definitions, which can be processed by the WebME system to fill requests. A class represents a Software Development Environment, like NASA/SEL. Direct and indirect attributes can be assigned to each class by an appropriate DDL statement. Direct attributes are values of external databases (received via data wrappers). Therefore, we have to specify a name, a unit, an interval (e.g., a weekly data collection), a host, a port and a binary (which reads values from the external database) for each attribute. Indirect attributes are combinations of direct ones (e.g., the difference of two direct attributes).

Requirements: (a) WebME allows for accessing different kinds of data repositories and visualization of therein stored data. Distributed development is supported by its mediated architecture. It is further possible to validate ingoing data by specifying data units and collection intervals for each attribute. WebME pro-

vides a set of standard data collection and processing techniques via its script language and supports their adaptation. Moreover, it is possible to combine different attributes via these mechanisms and build kinds of abstraction levels. The web-based architecture guarantees accessibility and up-to-date project information. (b) The integration of measurement paradigms and goal-oriented data visualizations are only partly addressed by the script language. (c) Packaging mechanisms are not addressed by the logical architecture.

5.6 PPM

Idea: The process performance manager (PPM) is a tool to support the management of business processes and was developed by IDS Sheer AG [18]. The aim is (1) to guarantee compliance with activities and effort plans, (2) to identify weak points in process execution performance, (3) to optimize the business process by identifying improvement potential, and (4) to assess the achieved improvements on the business process. Therefore, it provides a basis for decision making within an organization. The idea of PPM is to close the feedback gap between business process specification and execution. PPM provides functions for observing and assessing the performance of a current business process, and for providing feedback about it. Furthermore, 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. Through baseline specification, statistical analysis, and trend identification, PPM is able to identify deviations from baselines and to inform decision-makers.

Architecture: The architecture is shown in Fig. 8. Basically, it consists of four layers. (1) PPM is able to integrate existing source systems via an XML interface layer. An existing system accesses the PPM kernel, which runs as a server, via these adapters. (2) The PPM kernel layer includes a relational SQL database to store all relevant information of the source systems. In addition, the kernel includes a module to generate processes and compute the KPIs. (3) A user interface layer, which runs as a client, provides access to the PPM kernel and allows navigation of process data and visualization in the form of diagrams and tables. Therefore, a Java 2-compliant web browser is sufficient. (4) Finally, we have a layer with additional modules, like modules for process control, change management, analysis, and evaluation. The modules provide additional functionality to the PPM system.

Requirements: (a) The PPM system provides KPIs in order to assess the process performance and in addition a set of modules for further functionality, like process analysis. The system is adaptable to different projects through a variable XML adapter. Up-to-date project information is provided via a web browser. (b) PPM concentrates on business processes and does therefore not support distributed

software development in particular, but the architecture provides concepts to access different, distributed data sources via its XML adapter. Support for data validation and packaging project experience depends on the integration of additional modules. The same holds for integrating data from a reuse repository and data visualization mechanisms. (c) However, integrating a measurement paradigm and support for different data abstraction levels is not addressed.

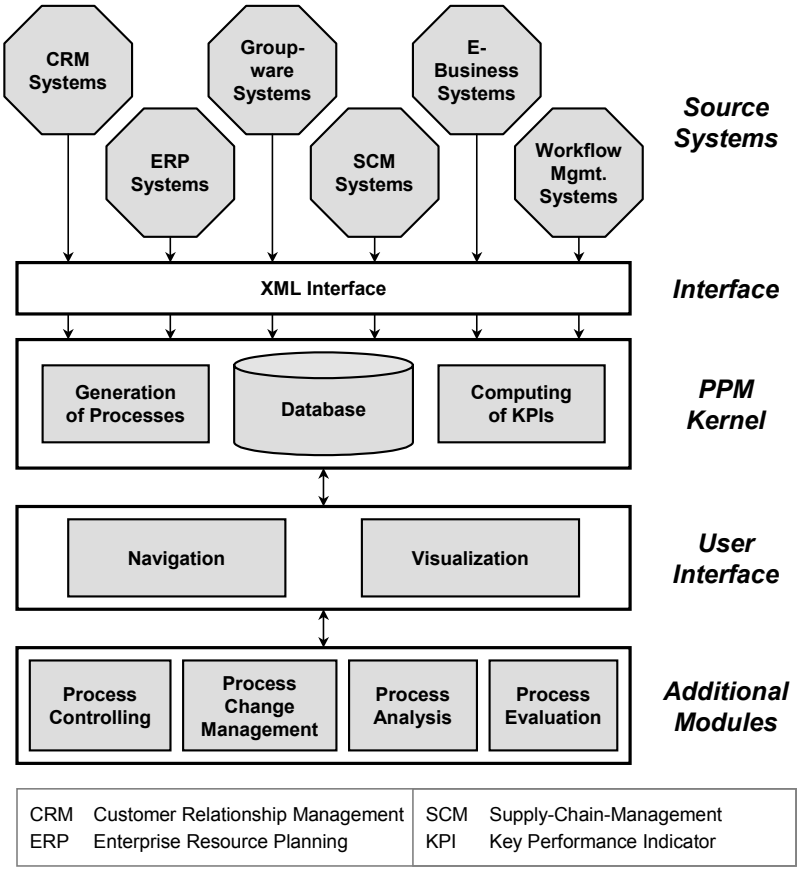

Fig. 8. Logical architecture of PPM [18].

5.7 PAMPA

Idea: PAMPA (project attribute monitoring and prediction associate) [35] is a tool that 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 a control and improvement cycle of a particular project. Intelligent agents reduce the overhead of data collection. They replace manual and subjective

data collection and analysis with objective procedures and allow a cost effective, automated solution for project control. Agents are responsible for data collection, data analysis, and report generation, and inform the project manager in case of plan deviations. Agents are generated by expert systems, which get their inputs from the PAMPA system.

Architecture: The basic architecture of PAMPA is shown in Fig. 9. Project information is stored in an object-oriented data management schema. PAMPA provides a set of predefined objects with relationships and attributes. The attributes correspond to measurement data or to nominal values of processes and products. They form the basis for further PAMPA functions. PAMPA uses Microsoft Windows and Office to visualize the collected data. In the area of data collection, several adjustments can be made to collect data from other environments, whereas visualization in the form of diagrams and tables and the generation of reports requires Microsoft Office.

Requirements: (a) PAMPA supports distributed software development and provides a set of control techniques through automated agents. Up-to-date information is provided through an MS Office front end. (b) PAMPA is integrated in a dual control and improvement cycle. However, mechanisms for integrating a measurement paradigm, packaging project data, or extending and adapting the set of applied control mechanisms are not fully addressed. (c) Concepts for accessing a reuse repository and data validation mechanisms are not addressed. The same holds for goal-oriented data visualization as well as data compression and abstraction mechanisms.

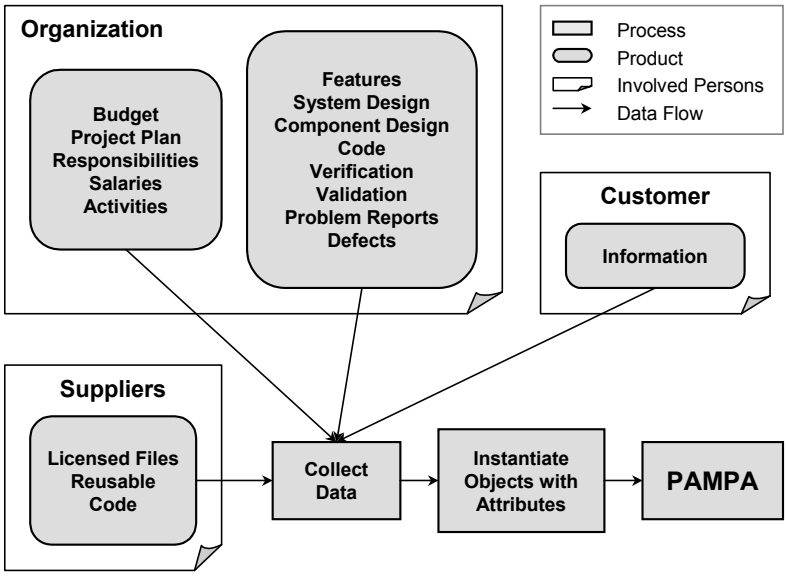

Fig. 9. Logical architecture of PAMPA [35].

5.8 SPCC Technology

Idea: The SPCC Technology approach was developed at the University of Kaiserslautern and the Fraunhofer Institute for Experimental Software Engineering (IESE) [14] [15]. The aim is to present the collected data in a goal-oriented way in order to optimize the measurement program and to effectively detect plan deviations. The benefits of this approach include (1) improvement of quality assurance and project control by providing a set of custom-made views on measurement data, (2) support of project management through early detection of plan deviations and proactive intervention, (3) support of distributed software development by means of a single point of control, (4) enhanced understanding of software processes and their improvement via measurement-based feedback, and (5) preventing information overload through custom-made views with different levels of abstraction.

Architecture: The architecture (see Fig. 10) is organized along three different layers. The information layer gathers all information that is essential for the basic functionality, for instance, measurement data from the current project, experiences from previous projects, and internal information, like all available purpose-oriented techniques and methods, the so-called SPCC functions. The functional layer performs all data processing activities, that is, it performs the currently used functions and composes role-oriented views. Finally, the application layer is responsible for all interactions with an user, that is, it provides the resulting information of the functional layer to a user and receives all incoming user requests. Each layer consists of several conceptual units, which provide the essential functionality.

Pool Management: The pool management unit accesses an expandable and generic control pool, which stores all elements needed to support project control. The first type of elements are so-called functions, which are able to apply 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 measurement goal, we need explicitly defined views, the second type of elements, of the processed data, such as one presentation suitable for the project manager, one for the quality assurer, and so on. Together, functions and views form a so-called visualization catena (VC). Functions process the measurement data, and views present the resulting information according to a certain measurement goal. The pool management unit is responsible for accessing the control pool, that is, it retrieves appropriate VC information and stores new, generalized functions and views in the corresponding sections of the control pool.

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 ex-

perience (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), validating incoming data, and integrating new experiences into the (organization-wide) EB. Therewith, the EB management unit provides all information necessary in order to perform the chosen functions.

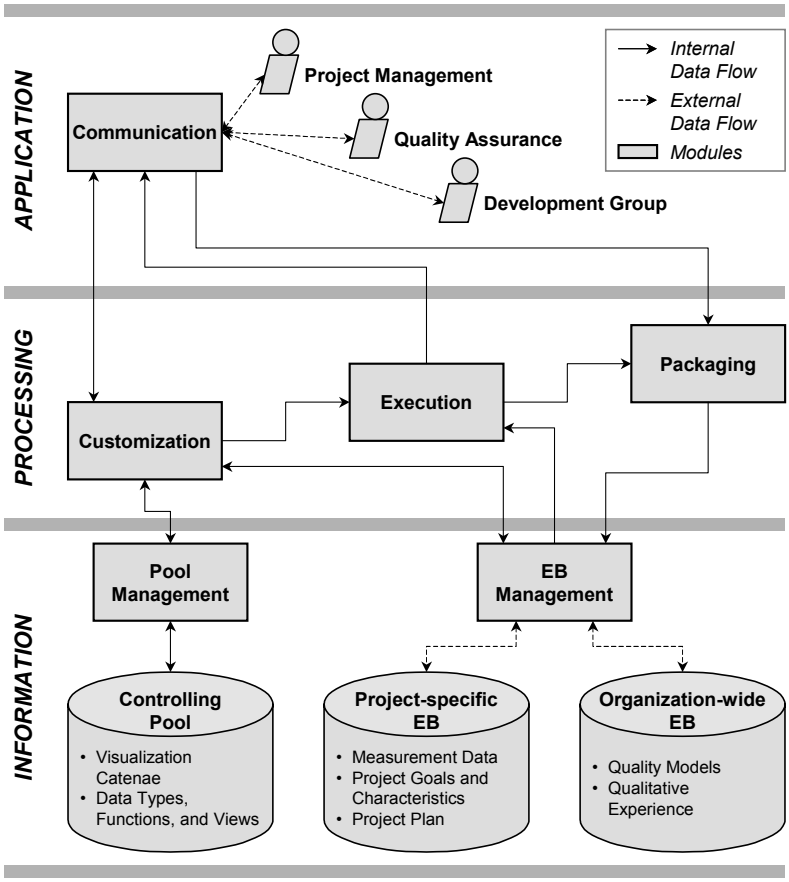

Fig. 10. Logical architecture of SPCC Technology [511].

Customization: The customization unit is the most complex conceptual unit. (1) At first, we have to initialize the EB; that is, we have to define all necessary data sources. (2) Afterwards, the pools are tailored to the project-specific needs; that is, we need to choose appropriate functions and views according to a certain usage or measurement goal (e.g., as part of 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 or views are defined (e.g., by the project administration), we need to generalize and integrate them into the respective pools for future usages.

Execution: The execution unit receives the chosen and adapted functions from the customization unit, determines input and output information, the function body, and the relationships with other functions. During execution of the chosen functions, the unit receives the respective input information from the EB management unit, respectively from a previously executed function. Furthermore, it receives the chosen and adapted views from the customization unit and determines the relationship between the views, and which function results have to be visualized by which view. All views are finally delivered to the user communication unit for device- or tool-specific visualization.

Packaging: The packaging unit summarizes all experiences gathered through the usage of SPCC Technology, generalizes them in order to be reused by future projects, and delivers them to the EB management unit for integration into the respective section of an experience base.

User Communication: First, the communication unit handles security issues, like the access granted to a specific user; that is, it permits a certain user to access the results of a certain set of functions, respectively a certain set of views. Second, it provides a graphical user interface (GUI) in order to customize the VC according to project goals and characteristics (via the customization unit). Third, it visualizes the views (delivered by the presentation unit) according to the chosen output interfaces.

The current SPCC Technology implementation covers the main aspects of the presented logical architecture with the exception of the packaging unit and a fully automated selection of functions and views according to a previously defined measurement goal. The control pool incorporates a set of control techniques and methods.

Requirements: (a) A user is able to access data and analysis results via the communication unit from different development locations. The formal VC definition allows for specifying explicit data validation rules. Moreover, a flexible and extendable pool of standard control techniques is provided and the specified VC is adaptable to different project contexts. A set of role-dependent views allows for goal-oriented data visualization and building up view hierarchies in order to visualize results of included data compression and abstraction mechanisms. The communication unit guarantees up-to-date information for every user. (b) A measurement paradigm (like GQM) can be used to formally derive a VC from measurement goals. However, currently no integrated mechanism is implemented. At present, the packaging module only exists conceptually; that is, data packaging depends on an SPCC function, which has to access a corresponding repository explicitly.

6 Integrated Techniques and Methods

In the following we give a few samples of techniques and methods integrated into the previously mentioned tool-based software project control approaches. The presented techniques support different roles within a software development project. Some are more technical-oriented (e.g., Classification Tree Analysis) and some are more management-oriented (e.g., Dynamic Variables). The aim of this section is to get an impression of the variety of integrated techniques and methods.

6.1 Classification Tree Analysis

Classification tree analysis is a widely-used statistical method that is used in the context of Amadeus to identify error-prone software components on the basis of previous software releases [30]. According to the 80:20 rule, 20% of a software system cause 80% of costs (because of error-proneness). Before we can start to build a classification tree, we have to define a so-called target class, for instance, all components with more than *n* interface errors. After that, a recursive algorithm searches metrics to distinguish between components inside and outside the target class based upon components of previous software releases. Therefore, a special metric is chosen to classify the components appropriately. The metric itself is selected with a specific function, which is beyond the scope of this article. 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.

Fig. 11 shows a classification tree example. The target class is defined by components with more than *n* interface errors. The first metric to classify the set of components is the number of data bindings (potential data exchanges via global variables). We get four different sets of components, namely, components with 0 to 3 data bindings, 4 to 5, 6 to 10, and more than 10 data bindings. Again, we select one metric for each node to partition the set of components within the node. This will be done until all components within one node are either inside or outside the target class. The first node contains all components with 0 to 3 data bindings. All these components are outside the target class and therefore, the termination criterion for this branch is met. The next node contains components inside and outside the target class. Therefore, we need another metric to classify them. The number of revisions is chosen to partition the set of components with 4 to 5 data bindings. According to Fig. 11, we get two sets that match the termination criterion for this branch of the classification tree. All components with 0 to 12 revisions are outside the target class and all components with more than 12 revisions are inside the target class. The resulting classification tree is presented in Fig. 11. With this technique, error-prone or high-risk components can be identified and treated with more attention.

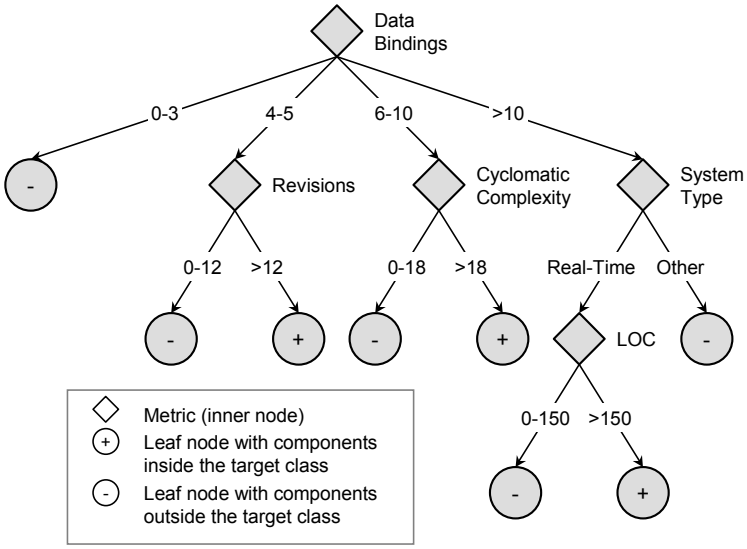

Fig. 11. Hypothetical classification tree [30].

6.2 Dynamic Variables

Doerflinger and Basili [11] describe the use of dynamic variables as a tool to monitor software development. The work was basically done in the context of the NASA SEL project and SME. 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 (synonymous to an observed project attribute) is generated from measurement data of one or more completed projects in order to compare projects in progress to 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.

To compare two similar projects, some points of time are needed to synchronize the comparison of variables. This can be *milestones*, such as the start or the end of a software development phase. But this is not the only crucial point. Variables, such as programmer hours and number of computer runs, are project dependent and therefore even hard to compare with other projects. We need to normalize the variables to compare values of different projects. The easiest way to create a project independent variable is to combine two dependent variables. The result is a relative measure, such as programmer hours per number of computer runs. Generally, there are two possibilities to express the time flow of a variable within a project. The first one is to measure the total number of events that have occurred from the beginning of the project up to the present (*cumulative*), and the

second one is to measure the number of events that have occurred since the last measurement of the variable up to the present (*discrete*). For example, let us assume four development phases with respective efforts e_1 to e_4 for each phase. These four values represent a discrete measurement, because we measure the effort of each phase separately. The corresponding cumulative approach measures the whole effort from the beginning of the project, that is, we get a total effort of e_1, $e_1 + e_2$, $e_1 + e_2 + e_3$, and finally $e_1 + e_2 + e_3 + e_4$ after each of the four phases, respectively. Both ways offer different analysis aspects and conclusions.

The introduced method uses one table for each relative measure (i.e., for each project independent variable). This table lists possible interpretations/causes for deviations above or below a baseline of the measure. An example is shown in Table 1, which presents possible interpretations for the deviation of a measure called *programmer hours per computer run*. The table is divided into a row for deviations above and below an appropriate baseline of the measure. Each interpretation has two columns with cross references to tables of other measures, which are represented by numbers. A reference means that the corresponding interpretation is listed in the referenced tables as well. For instance, the interpretation *high complexity* is also listed in tables with numbers 1, 2, 4, 8, and 9 in row *above normal*.

Table 2. Programmer hours per computer run [11].

Type	Interpretation	Above Normal	Below Normal
Above Normal	high complexity	1, 2, 4, 8, 9	
	modifications being made to recently transported code		9
	changes hard to isolate	4, 8, 9	
	changes hard to make	4, 9	
Below Normal	easy errors being fixed		5, 9
	error prone code	3, 4, 5, 6	2, 8, 9
	lots of testing		6

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 the associated tables for each flagged measure. (3) Count overlaps of possible interpretations; that is, count the number of emergences of a certain interpretation in every flagged table. (4) Determine the most probable interpretation, that is, the interpretation with most overlaps. For instance, if n measures are outside the tolerance range (above or below normal), we have n corresponding lists with possible interpretations. Then, the number of overlaps for each interpre-

tation is counted. If an interpretation *I* appears more often than an interpretation *J*, the former is more probable than the latter.

6.3 Cluster Analysis

Li and Zelkowitz [23] describe the use of cluster analysis for extracting baselines from collected software development data. This data is either 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. These points of time are assigned to four elementary software development phases: design, code/unit test, system test, and acceptance test (see axis of abscissae in Fig. 12). If you monitor a certain attribute (such as number of modules changed) over several projects you get one measure pattern for each project. Then, we can build a so-called *measure model* by averaging the measurement values within one set of measure patterns by computing the average value for each point of time of our 15-dimensional vector. This measure model refers to the expected behavior of a certain attribute as a function of time. 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 measure model of this set is computed by:

$(m_1,\ldots,m_{15})$, where $m_i = \dfrac{p_{1,i}+\ldots+p_{n,i}}{n}$ for $i \in \{1,\ldots,15\}$.

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 measure model of all measure patterns within one cluster. 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. In other words, each vector represents a point in a 15-dimensional space, and two points $P=(p_1,\ldots,p_{15})$ and $Q=(q_1,\ldots,q_{15})$ belong to the same cluster for a given ε if and only if:

$\sqrt{(p_1-q_1)^2+\ldots+(p_{15}-q_{15})^2} < \varepsilon$.

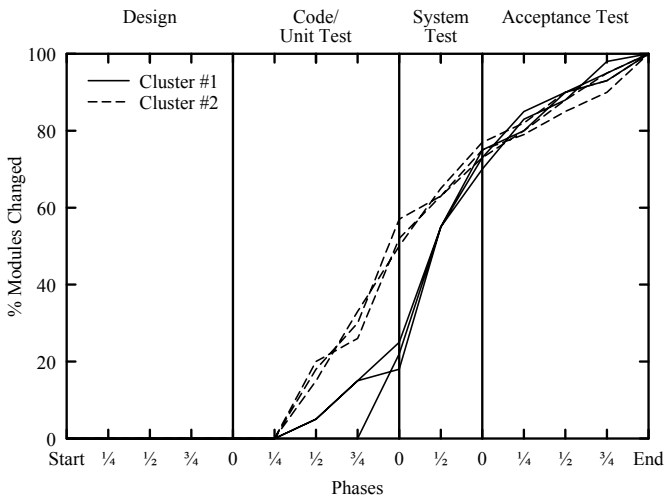

Fig. 12. Clustering of six measure patterns for modules changed [23].

Fig. 12 shows two clusters, built from the collected data for *modules changed* within six projects. Each measure pattern is represented by a line with 15 points. If the characterization vectors of these six projects were similar, they would be assigned to the same cluster in a static approach without computing their Euclidian distance. Cluster analysis allows us to build clusters for each considered variable of the six projects, which are based directly on the Euclidian distance of their measure models. Thus, two different clusters can be detected in Fig. 12. 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, he is able to determine general characteristics for a new project within a given cluster from the common characteristics of old projects within the same cluster. Clustering can also be used to detect relationships among measures or projects. If the clusters of two variables (e.g., reported changes and reported errors) consist of the same projects, this implies a relationship between the two variables. Vice versa, if two projects are within the same clusters for a significant number of variables, this implies a relationship between the two projects.

6.4 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 [39] 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. The EMA value e_i at time x_i is computed by:

$$e_i = (1 - \frac{2}{N+1}) \cdot e_{i-1} + \frac{2}{N+1} \cdot y_i \text{ for } i \in \{1,\ldots,15\} \text{ and } e_1 = y_1,$$

where y_i is the data value at x_i, $2/(N+1)$ is the smoothing constant, N is the number of points in the average, and n is the number of points in the scatter plot. The resulting 8-point EMA ($N = 8$) for an example scatter plot is presented in Fig. 13.

(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 differentiable. Therefore, another efficient way to compute the extreme values is chosen. The first step is to compute the so-called *instantaneous derivatives* or *delta values* of two consecutive points (x_{i-1}, e_{i-1}) and (x_i, e_i):

$$d_i = \frac{e_i - e_{i-1}}{x_i - x_{i-1}} \text{ for } i \in \{2,\ldots,n\}.$$

The delta values are the input for a second invocation of the EMA algorithm, mentioned previously. The resulting curve is called the *signal line* and is presented in Fig. 13. It represents the average slope of the instantaneous derivatives for the past N points. Therefore, if the signal line is zero in the neighborhood of x_i, you have a local maximum or minimum within the last N EMA values up to e_i. Such extreme values are called *pivot points*.

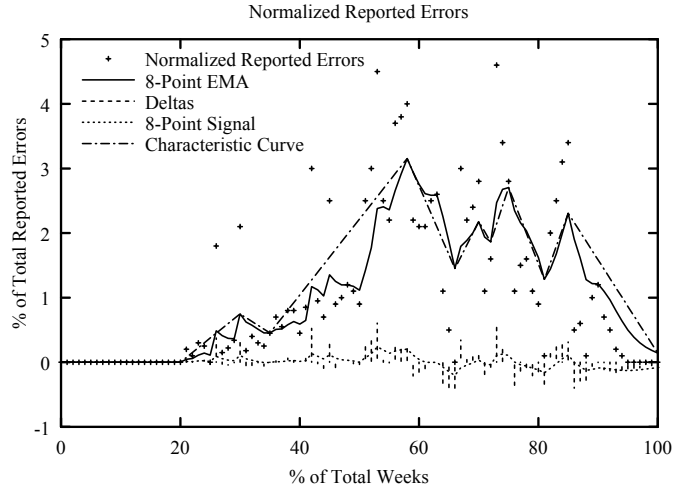

Fig. 13. Deltas, signal line and characteristic curve [39].

(3) The extreme values (called pivot points) of the smoothed scatter plot are the results of the second step of the algorithm. The last step is to connect each segment, defined by the pivot points, with a straight line. The resulting linear function is called *characteristic curve* and represents the major trends and trend changes of the original scatter plot (see Fig. 13).

6.5 Sprint I Control Approach

The Sprint I control approach [25] [26] [27] is a control technique used in the context of the SPCC Technology approach and built upon clustering algorithms to dynamically adapt the prediction of key project attributes during project execution. Sprint I is no pure control approach according to our definition because it predicts project attributes before project start and thus covers planning aspects as well. The prerequisite for a successful application of Sprint I to the currently performed project is that a software development organization has already 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, the context for each of these projects (i.e., the boundary conditions such as organizational, personal and technical constraints) needs to be characterized. The technique can be sketched as follows: First, the 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 its cluster curve (mean of all curves within a cluster) for predicting the attributes to be controlled. During the enactment of the project, the prediction is adapted based on actual project data. This leads to an empirical-based prediction and to flexibility for project and context changes.

The proposed control technique basically consists of five steps (see Fig. 14):

(1) Analysis. The first step of the technique analyzes the time-series of the measured attributes per completed project in order to get so-called characteristic curves of the attributes considered. A corresponding context is assigned to every characteristic curve, which represents the project environment that the curve originates from. A context description comprises all factors with a proven or assumed impact on the attribute values (such as people factors, technology factors, organizational factors, process factors).

(2) Clustering. For a certain attribute, clustering is used to identify groups of characteristic curves that belong together. These are, for instance, curves whose distance is less than a certain threshold. All characteristic curves within one cluster are averaged to get a model (the so-called cluster curve), which stands for the whole cluster. Again, a context is assigned to the aggregated curves based on a similarity analysis of the project contexts of the cluster.

(3) Initial Planning. During project planning, the relevant project attributes are estimated on the basis of cluster curves for the respective attributes of previous

projects. At the beginning of the project, no actual data are available and therefore, context characteristics must be used in order to find a suitable cluster curve for the project attribute under consideration (see Figure 1).

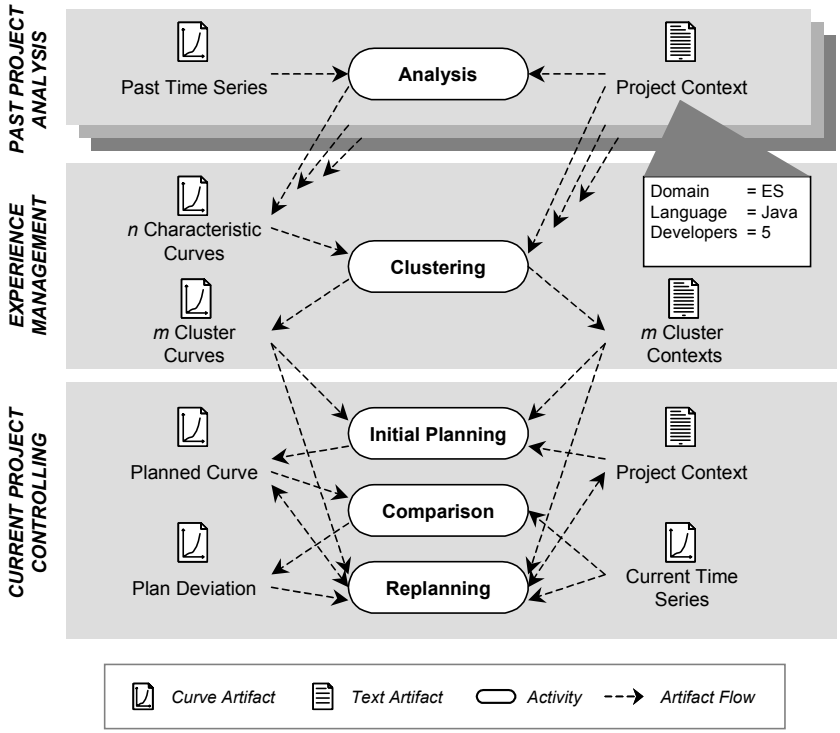

Fig. 14. SPRINT control approach.

(4) Comparison. During the enactment of the project, the current values of an attribute are compared with the predicted values of the cluster curve. If a plan deviation occurs, the predicted values have to be adapted with respect to the new project situation. Therefore, the distance between the two curves has to be computed regularly. If the distance is above (or below) a most tolerable threshold, project management has to be informed in order to initiate dynamic replanning steps, and a new cluster curve has to be sought in order to make a new prediction.

(5) Replanning. In case of significant plan deviation, the causes for the deviation have to be determined. We basically distinguish three different cases: The first one is that the experience we used to build our prediction model was wrong. The second one is that the characteristics we assumed for our project were wrong (e.g., the experience of the developers was low instead of high). In this case we have to adapt the project context. The third case is that problems occur in the project that lead to a change of the characteristics of the project (e.g., technology changed). In all three cases we can try to identify a new cluster curve within the set of computed clusters. Basically, there are three ways to choose a suitable clus-

ter for prediction: The first one is matching the contexts of the actual project and the cluster curves (like step 3). The second possibility is to use the current data of a certain attribute, which has been measured during the enactment of the project up to the present, and match it with the cluster curves in order to find the best cluster curve for prediction. This is a dynamic assignment approach, which incorporates actual project behavior. The third option is to combine the static and dynamic approach to get a hybrid one. If both possibilities lead to different clusters, a set of exception handling strategies can be applied and reasons can be sought. Afterwards, step 4 is iterated and uses the adapted prediction in order to further control the project.

7 Summary and Conclusion

This chapter conceptually defined the term Software Project Control Center to support project control for software development projects and integrated 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.

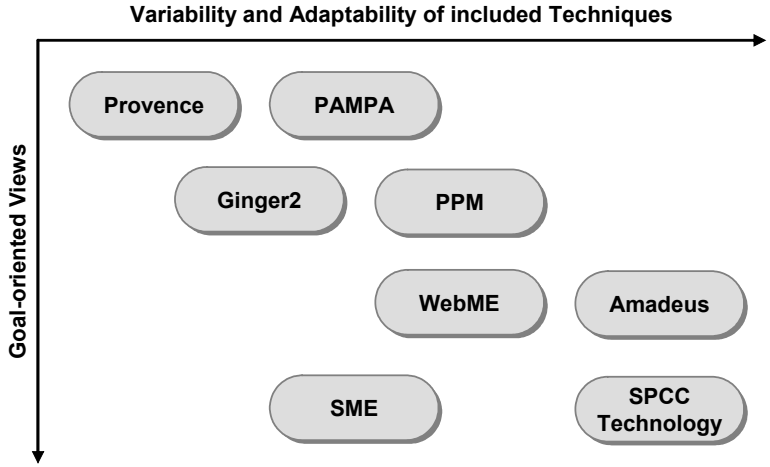

Fig. 15. Overview of discussed approaches.

Selected existing approaches for SPCCs in the context of the described framework were sketched. For that, different abstraction levels for each approach were considered. In particular, techniques, methods, and tools were associated with each approach. The focus of the description was on the logical architecture. We included the PPM approach in the field of controlling and assessing business processes as a representative for an approach outside the software development domain. Fig. 15 gives an overview of all discussed approaches and orders them ac-

cording to the scheme mentioned in Section 2 on the basis of their descriptions in Section 5. The first dimension (the columns) describes the degree of variability of techniques and methods included; the second dimension (the rows) describes the goal-orientation of the produced views.

Existing approaches offer only partial solutions for SPCCs. Especially goal-oriented usages based on a flexible set of techniques and methods are not comprehensively supported. This leads to future research directions:

An important research issue is the development of a schema for adaptable SPCC techniques and methods, 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. Both issues raise the question of an appropriate underlying logical architecture for an SPCC. The architecture should support (1) a variable and adaptable set of techniques and methods to provide the basic functionality, and (2) a goal-oriented presentation according to a certain usage goal of the SPCC. Furthermore, integration of popular visualization tools should be possible.

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 to 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 online 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 artifacts and other important project goals. An SPCC can be seen as a valuable contribution towards reaching this goal.

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. This work was partly funded by the Deutsche Forschungsgemeinschaft as part of the Special Research Project SFB 501 "Development of Large Systems with Ge-

neric Methods" and the Federal Ministry of Education and Research (BMBF) as part of the project "Simulation-based Evaluation and Improvement of Software Development Processes (SEV)".

References

1. V.R. Basili. Software Modeling and Measurement: *The Goal/Question/Metric paradigm*; Technical Report CS-TR-2956, Department of Computer Science, University of Maryland, College Park, MD, USA, 1992.
2. V.R. Basili. Applying the Goal/Question/Metric Paradigm in the Experience Factory. *Software Quality Assurance and Measurement: A Worldwide Perspective*; International Thomson Publishing: London, UK; Chapter 2, 1996.
3. V.R. Basili, G. Caldiera. Methodological and Architectural Issues in the Experience Factory. *Proceedings of the Sixteenth Annual Software Engineering Workshop*, SEL-91-006, 1991.
4. V.R. Basili, G. Caldiera, H.D. Rombach. The Experience Factory. *Encyclopedia of Software Engineering* 1: 469-476, 1994.
5. V.R. Basili, G. Caldiera, H.D. Rombach. Goal Question Metric Paradigm. *Encyclopedia of Software Engineering* 1: 528-532, 1994.
6. V.R. Basili, H.D. Rombach. The TAME Project: Towards Improvement-Oriented Software Environments. *IEEE Transactions on Software Engineering* 14(6): 758-773, 1988.
7. V.R. Basili, H.D. Rombach. Support for comprehensive reuse. *Software Engineering Journal* 6(5): 303-316, 1991.
8. V.R. Basili, D.M. Weiss. A Methodology for Collecting Valid Software Engineering Data. *IEEE Transactions on Software Engineering* 10(6): 728-738, 1984.
9. U. Becker, D. Hamann, J. Münch, M. Verlage. MVP-E: A Process Modeling Environment. *IEEE Software Process Newsletter* 10: 10-15, 1997.
10. L.C. Briand, C. Differding, H.D. Rombach. Practical Guidelines for Measurement-Based Process Improvement. *Software Process: Improvement and Practice* 2(4): 253-280, 1996.
11. C.W. Doerflinger, V.R. Basili. Monitoring Software Development Through Dynamic Variables. *Proceedings of IEEE Conference on Computer Software and Applications (COMPSAC)*: 434-445, 1983.
12. R.L. Feldmann, J. Münch, S. Vorwieger. Towards Goal-Oriented Organizational Learning: Representing and Maintaining Knowledge in an Experience Base. *Proceedings of the 10th International Conference on Software Engineering and Knowledge Engineering*: 236-245, 1998.
13. W.W. Gibbs. Software's Chronic Chrisis. *Scientific American* September: 86-95, 1994.
14. J. Heidrich, M. Soto. *Using Measurement Data for Project Control*. In Proceedings of the Second International Symposium on Empirical Software Engineering (Vol. II), Rome, October 2003, 9-10, 2003.
15. J. Heidrich. *Effective Data Interpretation and Presentation in Software Projects*. Technical Report 05/2003, Sonderforschungsbereich 501, University of Kaiserslautern, 2003.

16. R. Hendrick, D. Kistler, 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.
17. R. Hendrick, D. Kistler, 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.
18. IDS Scheer AG. *Optimieren Sie Ihre Prozess Performance – Process Performance Manager®*; IDS Scheer AG: White Paper, 2000.
19. G.E. Kaiser, N.S. Barghouti, M.H. Sokolsky. Preliminary Experience with Process Modeling in the Marvel Software Development Environment Kernel. Proceedings of the 23rd Annual Hawaii International Conference on System Sciences **II**, *IEEE Computer Society Press*, January; 131-140, 1990.
20. B. Krishnamurthy, N.S. Barghouti. Provence: A Process Visualization and Enactment Environment. Proceedings of the 4th European Software Engineering Conference, *Lecture Notes in Computer Science* 717; Springer: Heidelberg, Germany; 451-465, 1993.
21. L. Landis, F. McGarry, S. Waligora, R. Pajerski, M. Stark, R. Kester, T. McDermott, 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.
22. C.M. Lott. Process and measurement support in SEEs. *ACM Software Engineering Notes* 18(4): 83-93, 1993.
23. N.R. Li, M.V. Zelkowitz. An Information Model for Use in Software Management Estimation and Prediction. *Proceedings of the 2nd International Conference on Information and Knowledge Management*: 481-489, 1993.
24. F. McGarry, R. Pajerski, G. Page, S. Waligora, V.R. Basili, M.V. Zelkowitz. *An Overview of the Software Engineering Laboratory*; Software Engineering Laboratory Series Report SEL-94-005, Greenbelt, MD, USA, 1994.
25. J. Münch, J. Heidrich, 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, July 1-3, 2003.
26. J. Münch, 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.
27. J. Münch, 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.
28. J. Münch, J. Heidrich. *Software Project Control Centers: Concepts and Approaches*. Journal of Systems and Software, 70 (1), Elsevier, 2003.
29. Project Management Institute. *A Guide to the Project Management Body of Knowledge (PMBOKÆ Guide) 2000 Edition*. Project Management Institute, Four Campus Boulevard, Newtown Square, PA 19073-3299 USA, 2000.
30. A.A. Porter, R.W. Selby. Empirically Guided Software Development Using Metric-Based Classification Trees. *IEEE Software* 7(2): 46-54, 1990.
31. H.D. Rombach. Practical benefits of goal-oriented measurement. *Software Reliability*

and Metrics: 217-235, 1991.

32. H.D. Rombach, M. Verlage. Directions in Software Process Research. *Advances in Computers* 41: 1-63, 1995.
33. R.W. Selby, A.A. Porter, D.C. Schmidt, J. Berney. Metric-Driven Analysis and Feedback Systems for Enabling Empirically Guided Software Development. *Proceedings of the 13th International Conference on Software Engineering*: 288-298, 1991.
34. M. Shaw. Prospects for an Engineering Discipline of Software. *IEEE Software* 7(6): 15-24, 1990.
35. D.B. Simmons, N.C. Ellis, H. Fujihara, W. Kuo. *Software Measurement – A Visualization Toolkit for Project Control and Process Improvement*; Prentice Hall Inc: New Jersey, USA, 1998.
36. R. van Solingen, E. Berghout. *The Goal/Question/Metric Method, A Practical Method for Quality Improvement of Software Development*; McGraw-Hill: UK, 1999.
37. R. Tesoriero, M.V. Zelkowitz. The Web Measurement Environment (WebME): A Tool for Combining and Modeling Distributed Data. *Proceedings of the 22nd Annual Software Engineering Workshop (SEW)*, 1997.
38. R. Tesoriero, M.V. Zelkowitz. A Model of Noisy Software Engineering Data (Status Report). *International Conference on Software Engineering*: 461-464, 1998.
39. R. Tesoriero, M.V. Zelkowitz. A Web-based Tool for Data Analysis and Presentation. *IEEE Internet Computing* 2(5): 63-69, 1998.
40. K. Torii, K. Matsumoto, S. Kusumoto. *GINGER: A Quantitative Analysis Environment for Improving Programmer Performance*; Technical Report, Graduate School of Information Science, Nara Institute of Science and Technology, 8916-5 Takayama, Ikoma, Nara 630-01, Japan, 1995.
41. K. Torii, K. Matsumoto, K. Nakakoji, Y. Takada, S. Takada, K. Shima. Ginger2: An Environment for Computer-Aided Empirical Software Engineering. *IEEE Transactions on Software Engineering* 25(4): 474-492, 1999.