Fraunhofer Institut Experimentelles Software Engineering

A Systems Perspective on Software Process Improvement

Authors:
Andreas Birk
Dietmar Pfahl

Accepted for publication in proceedings of PROFES'2002

IESE-Report No. 047.02/E
Version 1.0
August 12, 2002

A publication by Fraunhofer IESE

Abstract

Software process improvement often lacks strong links to project management and control activities, which are concerned with identifying the need of process change and triggering improvement initiatives. Project management, on the other hand, often fails at selecting appropriate software engineering methods and technology that help to ensure project success. This paper proposes a model that guides project managers (1) to set up a project so that it can reach its specific goals and (2) to identify corrective actions (or changes) once a project is at risk of failing its goals. The model complements established improvement methods such as CMMI, GQM, and Experience Factory and links them to those project management activities that often are the starting point of improvement initiatives.

Table of Contents

1 Introduction

Today's software process improvement (SPI) methods offer little guidance for decision making on concrete improvement actions. For instance, when a project manager identifies a schedule overrun that threats timely product delivery, then improvement methods hardly give any specific recommendations nor guidance to get the project into schedule again. At the other hand, improvement methods are useful for emphasizing the role of software engineering methods and technology within a project, an aspect that conventional project planning widely neglects.

Instead of offering concrete problem solutions, improvement methods either guide organizations towards the identification of general improvement potential (i.e., benchmarking-based improvement) or help an organization to enhance its basic problem solving capabilities (i.e., feedback-based improvement). Examples of benchmarking-based improvement are ISO/IEC standard 9000:2000 [14] and the Software Engineering Institute's (SEI) Capability Maturity Model (CMMI) [10]. Examples of feedback-based improvement are the Experience Factory approach [2], measurement methods like Goal/Question/ Metric (GQM) [3][7][18][25], the SEI's PSM [11], and the Balanced Scorecard [15], as well as knowledge management approaches to SPI (e.g., project post mortems [16][5]).

Project planning focuses on aspects like deliverables, milestones, staff and other project resources, time, budget, risk, etc. Software engineering method and technology do usually not play a central role in project planning, although it can be crucial for project success to chose the right methods and to deploy them in the right way. For instance, an insufficient integration and testing strategy can easily make a project fail, even if all other project phases went extraordinary well.

This paper proposes a model that guides project managers (1) to set up a project so that it can reach its specific goals and (2) to identify corrective actions (or changes) once a project is at risk of failing its goals. Both aspects are equally relevant to project management and software process improvement. The model adds a focus of software engineering method and technology to project management. It also complements established software process improvement methods and grounds them stronger in core project management activities.

The model is defined as a systems model, which facilitates rapid and well-informed decision making. It also provides a framework for the detailed analysis of specific project phenomena, such as identifying root causes of schedule overruns and assessing the effects of adding new staff to the project. Throughout this paper, the model is denoted *SPI Systems Model*. This name is not fully appropriate, because we focus on the project management related aspects of SPI and do not address long-term organizational improvement activities in the first place. However, those long-term aspects of improvement are covered by most established improvement methods, and it will become clear how the proposed model integrates with them. For this reason we find it acceptable to stick with the simplifying but concise model name.

The sections of this paper briefly introduce the fundamentals of systems thinking (Section 2), present the SPI Systems Model (Section 3), and explain how the SPI Systems Model can be deployed by project management (Section 4). Section 5 discusses the presented approach with regard to project management and other SPI approaches. Section 6 summarizes the main conclusions of the paper.

2 Systems Thinking

In this paper we use systems thinking and cybernetics as a paradigm for modeling and discussing software project management and software process improvement. Systems models contain the main concepts of a phenomenon of interest (e.g., software projects) and describe how these concepts interact with each other (e.g., a project goal determines some aspects of a project plan, and an external event can impact the course of a project).

The usual paradigm for analyzing software projects is the process paradigm. It focuses on sequences of activities and addresses questions like "what is done when and by whom?". We want to take a different perspective: Our interest is not "what" is done, but "why" it is being done. For this purpose, systems thinking is a much better paradigm for analyzing and understanding the managerial, organizational, and socio-technical problems in software projects. It describes how *and* why systems behave the way they do.

Originating in the seminal work done by Norbert Wiener on cybernetics [28], until today there have probably been given as many definitions of systems thinking as there were scientists working in the field (e.g., Forrester [13], Checkland [8], Weinberg [27], van Bertanlaffny [4], etc.). In this paper, we follow the sufficiently broad but still concise definition given by Peter Senge who considered systems thinking the activity of contemplating the whole of a system and understanding how each part influences the rest [24]. In the case of a socio-technical system, like, for example, a car with a driver and passengers, this would include the analysis of how the actions of the individuals sitting in the car influence the behavior of the system.

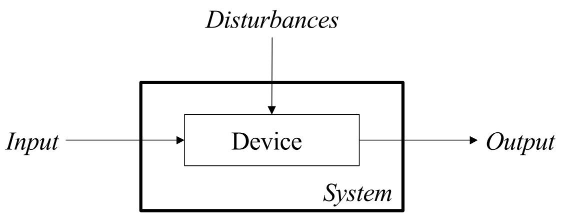

Figure 1: Open system without feedback

An important step toward systems thinking is to recognize that the internal structure of a system and the feedback processes that govern the relationships between system elements are the explanatory factors for its overall behavior rather than external disturbances. This way of looking at the source of system behavior requires that the system be considered essentially closed and not open. An open system (cf. Figure 1) basically is considered a device, e.g., a car *without* driver and passengers, that receives some input, e.g., the pressure on the gas pedal executed by the driver, and produces some output, e.g., the velocity with which the car moves. In a closed system (cf. Figure 2), e.g., a car *with* driver, again there is some input, e.g., the request or goal to reach the next town within 30 minutes, and some output, e.g., the velocity with which the car has to move in order to reach the goal.

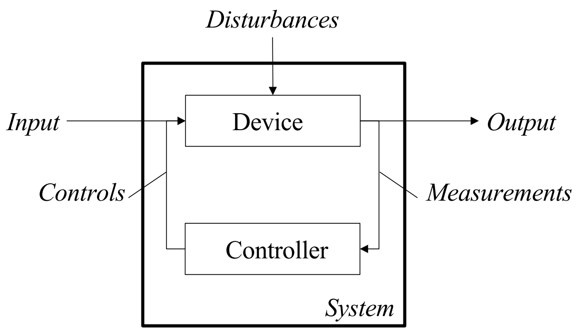

Figure 2: Closed system with feedback and control

In contrast to the case of the open system where the velocity of the car was dependent on some external influence, in the case of the closed system, the velocity of the car is controlled by the system itself. This happens through information feedback. By collecting measurement data, i.e. observing the speedometer and the clock, the driver (the controller in Figure 2) can calculate at any point in time how fast he must drive in order to achieve the defined goal. Based on measurements and some calculations, the driver decides whether he should change the pressure on the gas pedal (the control in Figure 2). It should be mentioned that in the case of the closed system, the controller would automatically take under consideration external disturbances (e.g. a steep hill) as long as the effect on the device is adequately reflected by the measurements – and neither misperceptions nor miscalculations occur.

3 A Systems Thinking Foundation of SPI

In the previous section we argued that systems thinking is the application of feedback control systems principles and techniques to managerial, organizational, and socio-technical problems. In this section, we will discuss further the assumptions and concepts that are important in the context of systems thinking, and – by using these concepts – we introduce the SPI Systems Model.

3.1 Control and Feedback

Control theory is based on the explicit premise that the change of a system is, or can be planned. Control is the process of ensuring that operations proceed according to some plan by reducing the difference between the plan (or goal) and reality. Control can only be exercised over the components internal to the system and cannot be affected upon the external environment. Using feedback mechanisms facilitates control over the system.

Feedback is concerned with the control of a mechanism (or device) on the basis of its past performance. It consists of procedures that determine deviations from plans and desired states and that indicate and execute corrective action regarding these deviations. This entails gathering data on the state of the output, searching for deviations from the plan, and adjusting the input based on the results of the output. It thus establishes a relatively closed system of causes and effects. It also reduces the risk of failure and the effect of residual complexity and ambiguity.

Both feedback and control presuppose planning, at least in the form of setting goals and performance levels, as plans furnish the baselines and standards of control. The pattern of goal seeking behavior exhibited by a system is then expected to stay true to the identified goal. The implicit and rather mechanistic assumption is that the plan or target does not change and that future conditions will remain identical to past conditions. In a change intensive environment these assumptions, and the resulting self-regulating mechanisms, clearly do not work and either a forward looking anticipation strategy or a double-loop feedback system must be employed.

Double-loop feedback offers a more sophisticated alternative that allows for the adjustment of the input variables to the process as well as the adjustment to plans that are used to dictate performance standards. The ability to respond to change and alter performance standards encourages adaptability and improves the chance of long-term survival. It also enables the control mechanism to benefit from most feedback data and avoid defensive routines to discredit suspect data.

Double-loop control requires long-term planning in designing the double-loop and will consume larger resources. It enables the system to become more adaptable and to do so more rapidly rather than bind itself to historical patterns. This adaptation means that the system is capable of long-term learning and continuous improvement in a search for greater efficiency. In contrast, single-loop feedback only focuses on the short-term adjustments during the duration of the control activity that will maximize the efficiency of the current product. Such improvements only apply to the current control loop and do not feed back into long-term changes to the overall process. In other words, lessons are neither learned nor retained.

The phenomenon that double-loop learning requires substantial investments which will pay-off only on a long-term time scale, while simple (isolated) single-loop feedback will only have local impact that is not sustained, is quite well reflected in the SPI literature by the duality of strategic management and project management. By offering an SPI Systems Model that has well-defined links to strategic management, we offer a perspective on organizational learning that puts the focus to the project as the heart of any sustained SPI program without de-coupling it from the strategic level.

3.2 The SPI Systems Model

Figure 3 presents the SPI Systems Model. The initiation of a software development project is triggered by business goals (BusG) from which the specific project goals (ProjG) are derived (arc 1). An example of a business goal would be: "We need to increase our market share in the database marked from 10% to 20% within the next two years". The project goals for the development of the next database release could then be: "Compared to the previous release, shorten lead time by enforcing concurrent engineering and reuse; at the same time, improve product quality by at least 20% and reduce development cost by 10%".

In order to make ProjG operational, a transformation into product goals (ProdG) and process goals (ProcG) is necessary (arcs 2). Typically, ProdG is associated with functionality and quality (i.e. functional and non-functional requirements), while ProcG is associated with time and cost (or effort). In our example, ProdG would relate to the planned increase of quality by at least 20%, while process goals would relate to the planned reduction of lead time and development cost. It should be noted that the definition of ProcG not only depends on ProjG but also on ProdG (arc 3). For example, the improvement of product quality might impose a change in the process that is going to be executed, say by increasing the inspection intensity (i.e., the number of inspections) during the design phase.

The joint set of ProdG and ProcG forms the starting point for developing ProjP (arcs 4). ProjP is the result of the planning stage and the central control for the development stage of the project. It can be seen as an instantiation of available process, product and resource models in the organization (yielding, for example, a Gantt chart, a resource allocation plan, and a high-level product architecture) that serves project management (ProjM) as an instrument to define and control development activities such as execution of processes and creation of product-related artifacts (arcs 5). The process situation (ProcS) can be determined based on observation (measurement) of project progress, i.e. activities that have been concluded, milestones that have been passed, resources that have been consumed, etc. The product situation (ProdS) can be determined based on observation (measurement) of certain characteristics (e.g., size, quality) of intermediate products and the final product. Because creation of intermediate and final products is inherently dependent on the execution of processes, and thus the current process situation, Figure 3 shows a synchronization link between ProcS and ProdS (arc 6).

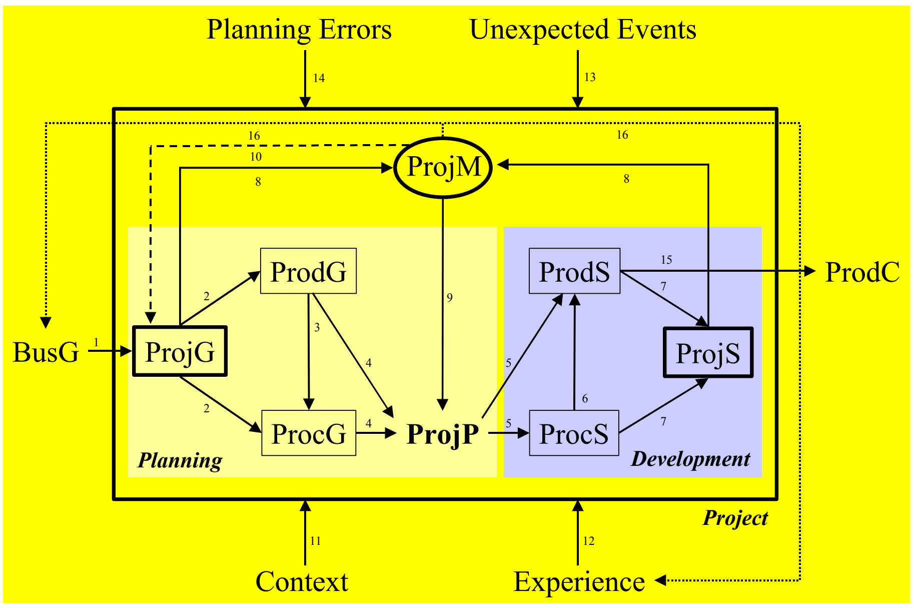

Figure 3: SPI Systems Model with single-loop and double-loop feedback

The project situation is a result of combining ProcS and ProdS (arcs 7). Based on measurement data and by using adequate models, e.g., static predictive models or dynamic process simulation models, a projection can be made until project end in order to facilitate comparison with ProjG for project control. This projection is labeled with ProjS in Figure 3.

Project management (ProjM) takes the role of the controller. Its main task is to compare ProjG with ProjS (arcs 8) and to initiate a change action if needed, e.g., when project deadline or product quality are at risk. There are two types of change actions possible that would create a single-loop feedback (arcs 9 and 10). The first case (arc 9), which can only induce a corrective change of ProjP without changing ProjG is labeled "single-loop feedback (inner loop)". The second case (arc 10), which may induce a corrective change of ProjG and, due to that, a change of ProjP, is labeled "single-loop feedback (outer loop)". Since it is not realistic to assume that ProjM can simply change ProjG without asking for and receiving the agreement from higher-level management, the outer loop feedback cycle is not fully endogenous to the system.

In addition to BusG, there are two important concepts that serve as an input to the system (i.e., the software project): Information and knowledge about the Context of the project (arc 11) and Experience about planning and development (arc 12). Context information is a relevant input to many planning and development activities and thus should be reflected in ProjP. Also Experience, e.g., best practices, should be taken into account when developing ProjP. Typically, Experience is available in the form of personal and implicit mental models, or it is made explicit and stored in an experience base in the form of quantitative and qualitative models (e.g., process models, product models, and resource models) that represent the past and current state-of-practice.

There are two major types of disturbances that impact the system: Unexpected Events (arc 13) and Planning Errors (arc 14).

The system output (arc 15), i.e., the project outcome, is the end product that will be delivered to the customer after project end (ProdC).

By extending the system boundaries and including strategic management (dealing with BusG) and experience management (dealing with Experience) into the system under observation, i.e., the software organization is the "device" on which the "controllers" strategic and experience management work, establishment of double-loop feedback becomes possible. In the extended model, double-loop feedback is possible in two ways (arcs 16). Either observations, lessons learned and best practices resulting from the project (and reported by ProjM) are used by strategic management (StratM) to alter BusG, or they are used by experience management (ExpM) to alter the models in the experience base.

It should be noted, that the structural similarity between Figure 2 and Figure 3 is not directly visible, because in Figure 3, apart from the project plan (ProjP), which represents the control, and the project management (ProjM), which represents the controller, no real world entities are depicted that would represent the device (or mechanism) on which control is applied. In the SPI Systems Model, the device is constituted by (1) the set of real world artifacts – besides

ProjP – ,i.e., process handbooks, standards and guidelines, development documents, technical documentation, test reports, user documentation, etc., and (2) all persons – besides ProjM – that assume a certain role within the project and their interaction. It can be argued, however, that project goals (ProjG), product goals (ProdG), and process goals (ProcG), are associated with the device during the planning stage, while the project situation (ProjS), process situation (ProcS), and product situation (ProdS) are associated with the device during the development stage. Inputs to the system are business goals (BusG), and – in a more indirect way – context information and experience from previous projects. Output from the system is the product that is eventually delivered to the customer (ProdC).

4 Deployment and Systems Thinking in SPI

This section illustrates how the SPI Systems Model can be deployed by project management. It addresses the following four kinds of deployment: (1) Goal-driven project planning, (2) project monitoring, (3) determination of change actions, and (4) performance of root cause and impact analysis by simulation.

4.1 Goal-Driven Project Planning

The key principle of the SPI Systems Model is that software development should be goal-driven, i.e., projects should be planned based on explicitly set goals. But how can this be done effectively?

First, it must be noted that the SPI Systems Model is not a process model. This means that one does not need to start with identification of project goals but can also start with an existing project plan and re-examine it in the light of relevant product and process goals. At the end it is important that the various goals and the project plan are consistent with each other. There are various ways through which this can be ensured.

Project goals are those that are raised by the project's customers and those set by the development company's internal authorities (e.g., product management or senior management). Each project goal should be written down together with information about who has raised it.

Product goals are related to functionality and quality of the product to be developed. They can also be related to cost or time. Table 1 contains several examples of product goals.

Product goal	**Goal category**	**Notes**
The software system shall have a maximum downtime of 6 h / year.	Quality	This is a Reliability goal, a typical kind of Quality goal.
It shall be possible to operate the system from web browsers (via HTTP) and from mobile phones (via WAP).	Functionality	Functionality is usually defined in the system requirements; the most important such requirements should be made explicit as product goals.
The first product release shall not cost more than € 1,2 Million.	Cost	Cost and Time can also be process goals. However, here they are clearly attributed to product.

Table 1: Examples of product goals

Process goals are related to project performance. They can (1) be related to cost and time, (2) address other project performance attributes (e.g., agility of the project organization, flexibility of the project processes, or work-based qualification objectives of the project staff), or (3) be derived from quality- and functionality-related product goals. Example process goals are shown in Table 2.

Process goal	**Goal category**	**Notes**
The total project budget until product release 1 shall be € 1,04 Million.	Cost	These Cost and Time goals are attributed to process. They are associated to similar product goals.
Project duration for stage 1 (pilot application) shall be four months.	Time	
It shall be possible to introduce additional browser compatibility requirements until two months before delivery of release 2.	Flexibility	Flexibility goals usually address the project's software development process and project organization.
The entire development staff shall gain experience in development and testing of internet software.	Staff Qualification	Process goals can also relate to human resources aspects of project performance.
System tests shall document that the system will have a downtime not longer than 6 h / year.	Quality	The Quality- and Functionality-related process goals are derived from the respective product goals.
The project shall develop the system for operation with web browsers (via HTTP) as well as mobile phones (via WAP).	Functionality	

Table 2: Examples of process goals

For defining a complete and consistent set of goals, a four-section grid can be used, which contains one section for each kind of goal: Initial project goals, product goals derived from project goals, process goals derived from project goals, and process goals derived from product goals. Each individual product and process goals should be checked for consistency with the other goals.

Once a consistent set of product and process goals is defined explicitly, a project plan can be developed that ensures that the goals can be fulfilled. For each goal, it should be possible to justify how the project plan helps attaining it. Project planning can be supported by repositories that document experience about how specific software engineering methods facilitate the achievement of certain goals (cf. [23][6]).

4.2 Project Monitoring

Project monitoring is the prerequisite for identifying the possible need for change and improvement actions. It consists of tracking product and process situation, and checking whether it is still likely that the respective goals can be attained. It is essentially based on software measurement. Therefore, indicators of goal attainment must be defined. They must allow for projecting the project situation at any given point in time to the expected situation at the end of the project. There are two basic strategies through which such project monitoring can be performed: Projection and milestone checkpoints. In practice, both strategies are usually combined with each other.

The projection strategy requires the identification of project indicators that can be identified (i.e., measured) in relatively small time intervals (e.g., on a weekly basis) to allow for sufficient projection quality. In addition, a projection function is required using which the current project indicators can be transformed into the expected project situation at project end time. Hence, in this case, monitoring is the comparison of a measurement-based estimation (e.g., derived from measurement of weekly staff effort) with the respective project goal (e.g., total effort budget of the project).

The strategy of milestone checkpoints breaks down the expected project situation at project end time into several project situations at major milestones. This requires some model from which the target values for each milestone can be derived (e.g., a model of typical effort distribution across development phases). The milestones are usually defined with time intervals of several weeks (e.g., each one or three months). Indicator measurement (e.g., measurement of accumulated staff effort) is required at least shortly before each milestone. In this case monitoring involves comparing the actual measurement data at each milestone with the previously defined target value for the respective milestone.

4.3 Determination of Change Actions

Change actions must be determined as soon as it occurs that the project is not likely to meet its goals. This can be due to two reasons: (1) The project plan is not appropriate for attaining the goals, or (2) the goals are not realistic. Both situations can happen because initial planning did not consider all relevant decision criteria, or because of the event of external changes that affect project goals or plan (e.g., customer changes strategy and wants the system to be developed on a different platform).

In the following, we will focus on the case that the project plan must be changed while the original goals can be kept. The other case (i.e., re-setting the goals) will not be considered here any further. It is widely similar to the initial project planning.

Figure 4 shows the input and output of project management activity ProjM (Determine Improvement Action), which establishes a feedback loop for project plan updates based on identified deviations of product and process situations from the respective goals. Input to the "Determine Change" activity are: Product goal and situation, process goal and situation, project plan (current status prior to change), context, and experience. Result and output of the Determine

Improvement Action activity is a process change decision, which leads to a change of project plan (ProjP).

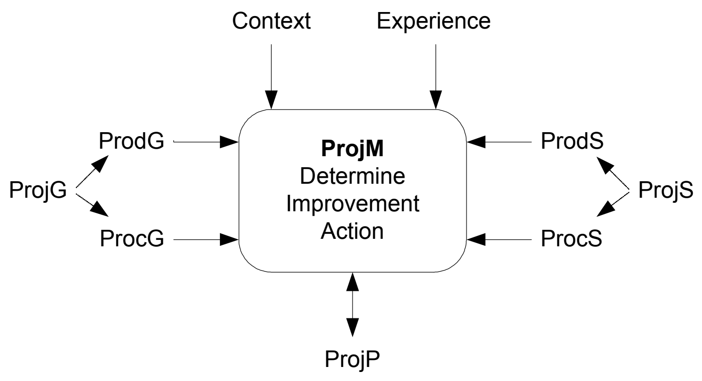

Figure 4: Input and output of the ProjM Activity (Determine Improvement Action).

Project managers can use this decision model for delineating their own individual decision making process, or for structuring a decision-making workshop with selected team members. In both cases it is useful when product and process goals are documented explicitly, an up-to-date project plan is available, and key indicators of the product and process status are known (cf. Section 4.2). In addition, information about relevant project context as well as experience about appropriate improvement measures for specific product or process goals will be helpful.

4.4 Performing Root Cause and Impact Analysis by Simulation

The SPI Systems Model provides a framework for the further refinement of model components and the relationships between them. In particular, the SPI Systems Model can be used as a blueprint for the simulation of project performance. Systematic application of simulation, in combination with measurement, can help uncover root causes of unexpected project behavior. It can also be used for evaluating planning alternatives and for performing impact analyses of proposed change actions prior to the actual implementation of the change [9][22].

The System Dynamics (SD) simulation modeling approach [12] closely follows the principles of Systems Thinking (cf. Section 2). Hence, SD is the recommended choice for building simulation models that represent and refine the SPI Systems Model [1][17][26]. SD simulation models focus on the formal (i.e., mathematical) representation of circular cause-effect structures that are held to be responsible for generating observed behavior of a system. Due to their flexibility and the possibility to combine hard data (empirical measurement) with soft data (beliefs and tacit knowledge), the SD approach allows for constructing adequate project models on different levels of detail according to the specific needs of project management. A methodology that systematically integrates measurement, quantitative modeling, process modeling, and project simulation using the SD approach has been presented in [19] under the name IMMoS (Integrated Measurement, Modeling and Simulation). Empirical evidence for the effectiveness and efficiency of the IMMoS methodology was collected in industrial case studies [20][21].

5 Discussion

The management of complex systems, like those typically underlying industrial software development processes, is very difficult. Based only on intuition and experience, it is generally not possible to comprehend the dynamic implications of so many interrelated loops carrying the system structure. If problems occur, their diagnosis is far from trivial. People often fail to think in terms of circular causal relationships and confound symptoms with causes. As a consequence, corrective policies implemented supply poor results for three main reasons: (1) The treatment of symptoms does not suppress the structural cause of the problem; (2) feedback systems resist policy changes because of internal compensation mechanisms; (3) the long term effects may be very different from short term effects, so that the implemented policy may actually worsen the problem in the long run.

For these reasons, we advocate the use of systems thinking in software process improvement. Established improvement methods implement systems thinking principles only to a limited extent. Several of these principles are often neglected. The SPI Systems Model presented in Section 3 aims at compensating this gap in established improvement methods. It can help leveraging the strengths of individual improvement methods and points out how specific improvement methods can be combined in order to receive maximum benefit for a software project or improvement program.

This section discusses the SPI Systems Model in the light of several improvement methods: CMMI-based SPI, measurement-based improvement using GQM, and the Experience Factory. Each method is compared with the SPI Systems Model, and integration possibilities of improvement method and the SPI Systems Model are outlined. The last subsection discusses appropriateness and justification of the proposed SPI Systems Model.

5.1 CMMI-Based Improvement

Improvement based on CMMI and other process assessment approaches (e.g., ISO 15504/SPICE or ISO 9001) compare (or assess) a project's or organization's software processes with a reference model of processes and evaluate the degree at which the assessed processes cover the reference model. Improvement suggestions can then be derived from the assessment results. However, the assessment methods do not include any specific recommendations on specific improvement suggestions or on the order in which possible process changes should be conducted.

From the viewpoint of the SPI Systems Model, assessment methods such as CMMI are a means for monitoring a project's process situation. They do not explicitly address any of the following concepts: Product-related aspects, project-specific process goals (i.e., other goals than those implicitly underlying the reference model of processes), specific decision making support for the identification of improvement suggestions, nor explicit support for specific project management activities. Even though it must be acknowledged that experienced process assessors usually take care of all these aspects when performing an assessment, the method itself does not address such issues.

5.2 GQM Measurement

Goal/Question/Metric (GQM) is a method for measurement and analysis in software engineering. Starting from the definition of project-specific measurement goals, appropriate measures (or metrics) are derived via a framework of question types. Usually, this is done by a measurement engineer, who acquires the needed information during interviews or group discussions with project team members. Afterwards, GQM addresses the preparation and execution of measurements and guides the analysis and interpretation of measurement results. Analysis and interpretation are usually performed in structured group discussions (so-called feedback sessions) of the project staff.

Concerning GQM's relation to the SPI Systems Model, GQM addresses the monitoring of both product and process situation with regard to individual project goals, offers a means for identifying improvement suggestions (i.e., the feedback sessions), and has been positioned as a tool for project management's monitoring and control tasks. Critics of GQM have argued that the approach is still too general, offering little specific guidance for standard measurement tasks. Likewise, it does not offer any specific decision making rules for the identification of improvement suggestions. In general, the implementation of software measurement can involve technical difficulties that make it not always easy to find a pragmatic approach to implementing measurement. For instance, it might take a relatively long time until measurement results are available and the first improvement suggestions can be made. From this viewpoint, GQM is one candidate solution (among others) for performing or supporting the measurement and control activities of the SPI Systems Model.

5.3 Experience Factory

The Experience Factory (EF) is a paradigm for experience-based (or learning-based) continuous improvement in software engineering. It builds on a cyclic process (the Quality Improvement Paradigm, QIP) of goal setting, planning, controlled action, and learning-based improvement. It also offers an organizational infrastructure that supports experience collection and deployment.

With regard to the SPI Systems Model, the EF is a conceptual framework that addresses most aspects of the SPI Systems Model. However, it does so on a relatively abstract level and does not offer operational guidance for the various tasks: The EF does not explicitly distinguish between product and process aspects, does not include specific monitoring and control mechanisms (EF implementations often use GQM for that purpose), and does not include specific improvement suggestions.

5.4 Appropriateness and Justification of the SPI Systems Model

The previous subsections have pointed out that the proposed SPI Systems Model complements established improvement methods by shifting focus on important project-related aspects of improvement. The SPI Systems Model includes key principles of systematic, feedback-based improvement: Explicit goal setting, systematic planning, informed decision making, and the need for accumulating experience (or best practice or patterns) about improvement actions that are appropriate within a specific given situation.

The importance of explicit goal setting and the separation of product goals from process goals have been emphasized by the PROFES improvement method [23]. Its relevance has been demonstrated in the PROFES application projects. The need for informed decision making, which should be supported by a sound understanding of the project situation and be based on accumulated past experience has been emphasized since the introduction of the Experience Factory. Recent contributions include the model-based simulation of software projects [19] and methods for knowledge management within SPI [6].

The SPI Systems Model formulates a project-based feedback system similar to GQM. However, it is not focusing on measurement alone and emphasizes the need for combined product and process monitoring. The SPI Systems Model also is linked to cross-project or organizational feedback (i.e., double-loop feedback) as formulated in the Experience Factory.

Concerning cross-project feedback, the SPI Systems Model addresses the core activity of identifying change actions. This is widely neglected in project management methods as well as in established SPI methods. For this reason, the SPI Systems Model guides project management's change activities and grounds established (organizational) SPI methods in software projects. It can be expected that such a link of project management and SPI helps overcome the still existing gap between both fields: Project management might gain a higher awareness of software engineering methods and technology, and SPI might easier attract project management's attention for the importance of long-term, sustained improvement activities.

6 Conclusion

This paper has introduced the SPI Systems Model that builds on explicit goal setting, separates software product from process, emphasizes monitoring of project state, and seeks for understanding the "why" of improvement needs and improvement actions. Anchor point of all these aspects is the software project plan, which transforms project goals into appropriate planned action. For this reason, the SPI Systems Model is grounded in project management: It views SPI as a tool that enables project management to keep a project in line with its goals.

The SPI Systems Model complements established improvement methods, which are usually not rooted in project management and lack guidance for the identification of concrete improvement suggestions. The model offers a pragmatic starting point for understanding how software project phenomena interrelate with each other, and why specific improvement suggestions might be superior to others in a given project situation. In cases where additional rigor and justification of decisions are needed, the SPI Systems Model can be refined and provide the basis for simulation-based root cause and impact analysis.

7 References

[1] Abdel-Hamid, T.K., Madnick, S.E.: Software Projects Dynamics – an Integrated Approach. Prentice-Hall (1991)

[2] Basili, V.R., Caldiera, G., Rombach, D. H.: Experience Factory. In: Marciniak, J.: Encyclopedia of Software Engineering, Vol. 1, pp. 511-519, Wiley (2001)

[3] Basili, V.R., Caldiera, G., Rombach, H.D., van Solingen, R.: Goal Question Metric (GQM) Approach. In: J. Marciniak: Encyclopedia of Software Engineering, Vol. 1, pp. 578-583, Wiley (2001)

[4] von Bertalanffy, L.: General Systems Theory, Foundations, Development, Applications. Georges Braziller, New York (1968)

[5] Birk, A., Dingsøyr, T., Stålhane, T.: Postmortem: Never leave a project without it. IEEE Software, 19(3), pp. 43-45, (2002)

[6] Birk, A.: A Knowledge Management Infrastructure for Systematic Improvement in Software Engineering. PhD Theses in Experimental Software Engineering, Vol. 3, Fraunhofer IRB, Stuttgart, Germany (2001)

[7] Briand, L.C., Differding, Ch., Rombach, H.D.: Practical Guidelines for Measurement-Based Process Improvement. Software Process Improvement and Practice 2 (4), pp. 253-280, (1996)

[8] Checkland, P.: Systems Thinking, Systems Practice. (1981)

[9] Christie, A.M.: Simulation: An Enabling Technology in Software Engineering. In: CROSSTALK – The Journal of Defense Software Engineering, pp. 2-7 (1999)

[10] CMMI Product Team. Capability Maturity Model Integration (CMMI), Version 1.1. Software Engineering Institute, Pittsburgh, PA (2002)

[11] Florac, W.A., Park, R.E., Carleton, A.D.: Practical Software Measurement. Software Engineering Institute, Pittsburgh, PA (1997)

[12] Forrester, J.W.: Industrial Dynamics. Productivity Press, Cambridge (1961)

[13] Forrester, J.W.: Principles of Systems. Productivity Press, Cambridge (1971)

[14] International Organization for Standardization: ISO 9001:2000: Quality Management Systems - Requirements. International Organization for Standardization (2000)

[15] Kaplan, R.S., and Norton, D.P.: The Balanced Scorecard: Translating Strategy into Action. Harvard Business School Press, Boston (1996)

[16] Kerth, N.L.: Project retrospectives: A handbook for team reviews. Dorset House, New York (2001)

[17] Lin, C.Y., Abdel-Hamid, T.K., Sherif, J.S.: Software-Engineering Process Simulation Model (SEPS). In: Journal of Systems and Software 38, pp. 263-277 (1997)

[18] van Latum, F., van Solingen, R., Oivo, M., Hoisl, B., Rombach, D.H., Ruhe, G.: Adopting GQM-based measurement in an industrial environment. IEEE Software, 15(1):78–86 (1998)

[19] Pfahl, D.: An Integrated Approach to Simulation-Based Learning in Support of Strategic and Project Management in Software Organisations. PhD Theses in Experimental Software Engineering, Vol. 8, Fraunhofer IRB, Stuttgart, Germany (2001)

[20] Pfahl, D., Lebsanft, K.: Knowledge Acquisition and Process Guidance for Building System Dynamics Simulation Models. An Experience Report from Software Industry. In: International Journal of Software Engineering and Knowledge Engineering 10, 4, pp. 487-510 (2000)

[21] Pfahl, D., Lebsanft, K.: Using Simulation to Analyse the Impact of Software Requirement Volatility on Project Performance. In: Information and Software Technology 42, 14, pp. 1001-1008 (2000)

[22] Pfahl, D., Ruhe, G.: System Dynamics as an Enabling Technology for Learning in Software Organisations. In: 13th International Conference on Software Engineering and Knowledge Engineering. SEKE'2001. Knowledge Systems Institute, Skokie, IL, pp. 355-362 (2001)

[23] The PROFES Consortium: PROFES User Manual. Fraunhofer IRB Verlag, Stuttgart, Germany (2000)

[24] Senge, P.M.: The Fifth Discipline – the Art & Practice of the Learning Organization. Doubleday, New York (1990)

[25] van Solingen, R., Berghout, E.: The Goal/Question/Metric Method: A practical guide for quality improvement of software development. McGraw-Hill, London (1999)

[26] Waeselynck, H., Pfahl, D.: System Dynamics Applied to the Modelling of Software Projects. In: Software Concepts and Tools 15, 4, pp. 162-176 (1994)

[27] Weinberg, G.M.: An Introduction to General Systems Thinking. Wiley, New York (1975)

[28] Wiener, N.: Cybernetics. Wiley, New York (1948)

Document Information

Title:	A Systems Perspective on Software Process Improvement
Date:	August 12, 2002
Report:	IESE-Report No. 047.02/E
Status:	Final
Distribution:	Public