Experiences and Results from Establishing a Software Cockpit at BMD Systemhaus

Stefan Larndorfer, Rudolf Ramler
Software Competence Center Hagenberg
Softwarepark 21, 4232 Hagenberg, Austria
{stefan.larndorfer, rudolf.ramler}@scch.at

Clemens Buchwiser
BMD Systemhaus GMBH
Sierninger Straße 190, 4400 Steyr, Austria
buchwiser@bmd.at

Abstract— What is the degree of completion of the current iteration? Are we on track? How accurate are estimates compared to actual effort? Software cockpits (software project control centers) provide systematic support for answering such questions. Therefore, like a cockpit in an aircraft, software cockpits integrate and visualize accurate and timely information from various data sources for operative and strategic decision making. Important aspects are to track progress, to visualize team activities, and to provide transparency about the status of a project. In this paper we present our experiences from implementing and introducing a software cockpit in a leading Austrian software development company. The introduction has been supported by a small-scale process improvement and a GQM-based measurement initiative. Furthermore, the paper discusses challenges and potential pitfalls in establishing software cockpits and shares some lessons learned relevant for the practitioner confronted with introducing a software cockpit in an industrial setting.

Keywords - software cockpit, measurement, GQM, dysfunction

I. INTRODUCTION

A software project can be a challenging endeavor that involves many critical aspects to be considered. One crucial factor is a motivated and closely collaborating team focused on the goal of developing software products that meets business goals with given resources and on time. Therefore, it is essential to closely track the progress of the project, to periodically evaluate the status of the product with objective measures, and to make reliable estimates about how much effort is required to complete the planned functionality.

Software cockpits (aka project dashboards or software project control centers [16]) are a means to provide the necessary systematic support for project control and quality management. A software cockpit is similar to an aircraft cockpit as it integrates relevant information for operative and strategic decision making [4, 5, 15]. Additionally, software cockpits contribute to more transparency in the software development process, increase team awareness [5], and help to improve estimation capabilities [13].

This paper describes how a software cockpit has been successfully set up and applied within a major Austrian software development company specialized on business software, BMD Systemhaus GmbH. Both, technical results and insights on important practical issues associated with the implementation and introduction of software cockpits in a real-world environment are reported.

Throughout the last few years the increased importance of tool support for project control and quality management resulted in a number of open source and commercial solutions (e.g. Pentaho, Qlikview), which are used in practice to build company-specific cockpits [9]. Still, however, the experience reported in the literature about establishing software cockpits in an industrial setting is very limited [5, 7].

The contributions of this paper are a report about establishing a software cockpit leveraged by a small-scale process improvement and a GQM-based measurement initiative. Furthermore, the paper discusses challenges and potential pitfalls in establishing software cockpits with emphasize on the often neglected socio-psychological issues concerning the provision, analysis and presentation of software repository data. Finally, some lessons learned relevant for the practitioner confronted with introducing a software cockpit are presented.

Hence the paper is structured as follows. Section 2 explains the project’s context. Section 3 gives an overview of the process used to develop and introduce the software cockpit. Preliminary results and the state of the cockpit at the time of writing are presented in Section 4. Section 5 lists the main challenges faced during the introduction. In Section 6, the paper is concluded with a summary and our lessons learned.

II. PROJECT SETTING

BMD Systemhaus GmbH (http://www.bmd.at) is a leading software development company for business software located in Austria. Since it’s foundation in 1972 the number of employees has grown to more than 300 with a rapid increase particularly in the last few years. The highly successful product is applied in more than 14.000 companies with a total of about 50.000 users.

BMD’s software product consists of a set of modules such as financial accounting, costing, payroll, ERP, CRM. The module structure provides the organizational frame for development. The teams are specialized on the different modules and combine domain expertise with software engineering know-how. The development of the entire product is organized on the basis of an iterative software development process with monthly releases that integrate the contributions of the specialized teams.

The comprehensive customer support department ensures that customer requests, ideas and defects related to the software product are analyzed and transformed into change requests and new requirements managed with the corporate-wide task and issue tracking system. This system has evolved to a central instrument to plan, monitor and
control software development. Altogether several different roles use the task and issue tracking system to contribute to the development of the software product.

- The **support team** records the requests of customers. A close contact to the customers helps to understand the customers’ desires and to distill appropriate software product requirements.
- The **developers** derive development tasks from product requirements, provide effort estimates, set up preliminary lists of tasks and a proposed scheduling of the tasks to future iterations.
- An **expert group** consisting of team leaders, product managers and sales managers assess the preliminary task lists via a business value analysis. The result is a task list aligned with the company’s business strategy. This task list is the input for the development teams.
- The development **team leaders** prepare the plans for upcoming iterations. They assign the existing tasks to developers, balance the overall workload with the available resources, and try to accommodate additional incoming requirements and long-term maintenance activities.
- The **management** directs the evolution of the software product across all teams in terms of functionality, system architecture and applied technologies. Currently several existing modules are moved to a new technological basis in parallel to the ongoing flow of development.

The dynamic nature of iterative development involving contributions from several hundred people working in different organizational units keeps the software product in a constant state of flux and makes it increasingly hard to keep track of the progress within and across iterations. The concept of software cockpits that integrate in-process metrics about the current status and progress of an iteration were identified as potential aid and a project to establish such a software cockpit was launched.

Following general goals were set at the beginning of the project:

1) Increase the transparency of the software development process by integrating data from the issue repository, time recordings and data from first level support.
2) Improve the ability to plan and predict software development. Provide precautionary information in case of schedule slips.
3) Provide timely and accurate information for all roles involved in software development and establish an institutionalized way to monitor and track progress across all development teams.

Transparency of the software development process means to show daily updated views that answer the following questions: What is being developed in the various development groups? What is planned for upcoming iterations? What degrees of completion have individual work packages?

Integrating data aims at providing a data basis that is able to answer cross-repository relevant questions. For example: How much time was needed for the implementation of a specific requirement and how much code had to be changed?

### III. Process Used to Establish the Cockpit

The implementation and introduction of the software cockpit followed a structured process encompassing nine overlapping phases with frequent feedback cycles. In the literature several processes and process frameworks for related initiatives such as the introduction of measurement programs, applying QM, or process improvement initiatives offer valuable input and guidelines also applicable for establishing software cockpits (e.g. [10, 3, 17, 9]). Furthermore, Heidrich et al. [12] proposed a process specific for the setup and usage of custom-tailored software cockpits. Our approach has been derived from the process of Heidrich et al., although several adaptations to meet the specific requirements of BMD and to include aspects of other examined processes resulted in the alternative version illustrated in Figure 1. In our case, the process can be divided in three major stages, which are (a) the definition and implementation of the software cockpit, (b) its evaluation, and – finally – (c) the rollout to the whole company.

In the first stage, emphasis was set to the analysis of existing processes and data in order to define new processes optimized for the cockpit’s requirements about data collection and measurement. The key question was: “How to establish the software cockpit to maximize the benefits for developers, team leaders and management?” Hence, from the beginning it was clear that this target would not be achievable without improving existing processes and associated repositories.

The evaluation of the cockpit involved two representative development teams that served as pilot groups throughout the whole project. These two teams were also the main contact during the analysis of the current processes and for experimenting with new processes or tools. The fruitful relation to the pilot groups brought about many valuable insights into the developers’ personal workflows and helped to draft an optimal user-centered software cockpit.

The next paragraphs will shortly describe each of the phases in establishing the software cockpit.

1) **Analysis of the current process:** In the first phase, the current development process has been analyzed and documented by observing and interviewing developers and team leaders from all different development teams. A detailed understanding of how the process is applied including variations between the different teams turned out to be important to develop a concept for the software cockpit suitable for all development teams.

In this phase it was also important to clarify further project constraints together with the management including specific standards or regulations that have to be fulfilled, company-specific policies or strategic decisions to adhere to, regulations concerning the processing of personal data, and the reporting requirement regarding the works council.

2) **Definition of a GQM model:** The GQM (Goal-Question-Metric) approach [2] defines a systematic technique to construct goal-centered measurement programs. Together with management and team leaders a GQM model consisting of organizational goals, related questions, and required metrics was developed. The model contained questions that could not be answered without changing the actual processes in order to collect the
necessary data. Thus, the GQM model was an important input for the improvement of the existing processes.

3) Elicitation of requirements for the target process: The results of the process analysis and the GQM analysis laid the foundation to elaborate the requirements for the new target process. The goal of this phase was to identify these requirements and characteristics independent from specific tools or approaches. The result of this phase was a document consisting of 16 short paragraphs that explains these requirements and considers role specific impacts on processes.

The requirements were discussed with the different teams to create awareness and to collect feedback as well as approval. It turned out that the 16 requirements served very well as a generally understood and accepted “manifesto” useful to communicate the rationale for the necessary process changes.

4) Definition of the target process: Based on the requirements identified in the previous phase, the detailed process definition was documented in the notation of evident-driven process chains. Thereby, the existing ISO 9001 process documentation was used as template. The process definition was mainly used as a reference for the people implementing the new processes, while in the communication with all other roles the process requirements written in natural language was preferred over the more complex and detailed process diagrams.

5) Development of tool support: The focus of this phase was on how the target process can be supported by existing and new tools. Effective and efficient tool support is essential to reduce the overhead and to obtain the acceptance of the busy developers and team leaders. Thus, for example, the existing issue task and tracking system has been extended to record additional data required for the target processes in new fields. Furthermore, some of the intended process changes can be illustrated very vividly to developers and team leaders by presenting a sequence of screenshots.

The first prototype implementation of the software cockpit was a result of this phase. The implementation has been based on existing, customizable and extendable dashboard solutions. This flexibility allowed to evolve the software cockpit in a series of small development and feedback cycles in close cooperation with team leaders and developers.

6) Launch of processes and cockpit in pilot groups: The new processes and tools were evaluated in a pilot phase of about four month. Thus, the two pilot groups involved since the very beginning of the project applied the proposed processes and the prototype implementation of the cockpit in development. Therefore, the pilot phase was started with a kick-off workshop. In the workshop the role of the pilot groups was clarified and the new processes and the intended application of the new or changed tools were introduced. The use of a clear language and unmistakably notions to convey crisp messages understood throughout the organization was an important aspect of this phase.

7) Continuous improvements with pilot groups: Direct observation and personal interviews with key users were applied to check whether the processes and tools were put into practice in the intended way and provided the expected benefits. Additionally, the entries in the software repositories was checked on a regular basis to make sure that the data collected in the new processes fulfilled the quality requirements and could be evaluated the intended way. Based on the gathered feedback, small adjustments to improve the processes and tools were made.

8) Evaluation of pilot phase: Are new processes and tools mature enough? Does the software cockpit provide the expected benefits for developers and team-leaders? By means of a structured interview, the members of the pilot groups evaluated the software cockpit. The resulting evaluation report was the basis for the positive decision to rollout the cockpit to the rest of the organization.

9) Company-wide rollout: The start of this phase was marked by an official announcement in the company’s newsletter and magazine proclaiming the introduction of the software cockpit along with the new processes and tools throughout the whole organization. In a meeting of the management team and the managing director the rollout strategy had been defined. To introduce the software cockpit and to disseminate the details the new processes and tools, presentations for all teams and departments were organized within a period of another four months.

IV. RESULTS

The introduction of the software cockpit brought about three major results: (1) a reorganized central repository for
software development artifacts such as requirements, bug reports, and development tasks; (2) an improved process for iterative software development; and finally (3) a software cockpit implementation (see Figure 2) capable to support the improved processes and based on the central repository. The cockpit implementation shows a number of characteristics considered essential for the effective application by developers and team leaders.

• A user-centered design aligned with the users’ daily activities keeps overhead at a minimum. All relevant information is presented with simple graphics [11] on a single screen. It can be personalized in terms of user specific views and filters.
• Visualized charts and metrics are easy to interpret and the presented data can be traced back to the individual activities in software development. The charts are designed to reflect the progress over time and allow comparing actual data with data from previous releases [14]. Abstract metrics and high-level indicators have been avoided as well as automated interpretation of the data in terms of “good” and “bad”. As a consequence, dashboard elements such as traffic lights or tachometers were omitted.
• Interactive analysis of measurement data [8] allows drilling down from aggregated measurements to individual data records (see Figure 4). In-place exploration is supported by mechanisms such as stacked charting of data along different dimensions (see Figure 3), tooltips providing details about the data points, and filters to zoom in on the most recent information.
• A progress bar shows detailed information about the degree of completion of the current iteration (Figure 4). Progress is calculated from estimates (upper bold bar) of resolved and planned tasks. The estimates are compared to actual values (lower bold bar) “booked” in the time recording system. Again, a drill down to individual tasks (lower bar charts) is possible.
• Problems with data quality are indicated as grey areas in the bar charts. In Figure 4, for example, estimates for tasks are missing and had to be represented with average values.

The project of implementing and introducing the software cockpit lasted about 1.5 years. About 100 developers and 120 employees from support contribute data to the cockpit. So far, about 4,500 entries have been recorded with the new processes and tools and are managed with the software cockpit. The log file analysis reveals that the cockpit is used every working day by an average of about 1.2 people. Throughout the four months of the pilot phase, the cockpit recorded more than 2,500 page hits.

The software cockpit was evaluated by developers, team leaders and management at the end of the pilot phase. The evaluation, which was based on structured interviews, showed that the involved people are very confident about the positive long-term effects of the software cockpit. A number of benefits for software development have been realized. The cockpit motivates a consistent application of the defined processes throughout all development groups and provides immediate feedback about data quality and consistency. It, furthermore, provides transparency about the current status of development on objective measures and allows forecasts on the workload for the next four to eight weeks.

Naturally, there has been some resistance to process changes and a critical discussion about the benefits of a software cockpit at the beginning of the project. The following concerns have been raised: (1) Too much transparency may put pressure on developers and may corset them in their way how they get their work done. (2) Emphasize on process compliance and the permanent need to provide the necessary measurement data causes too much overhead for developers in an environment where instant feedback to customer requests and new opportunities is essential. (3) The available development tools used, e.g., for issue tracking, building, and version control, do not optimally integrate and, thus, do not support the effortless collection of measurement data.

However, the evaluation confirmed that the developers appreciated the benefits of the software cockpit and accepted the inconvenience of data collection in order to improve the software development. Especially the first two concerns vanished as developers accustomed to the improved processes and the software cockpit fostered an attitude towards transparency, engagement and a culture of organizational learning and sharing of experience.
usage of measurement data: concerns the interaction between the organization and the particular surface a socio-psychological dimension that of technical nature. A number of different challenges in repositories poses many challenges, most of which are not organization with established processes, tools and facts make the contribution and value of software perceived as surveillance, but as presenting facts. These of the answers in the software cockpit (Figure 2) is not necessary events have to be measured when asking "reasonable transparency where necessary". Only necessary events have to be measured when asking questions like “Which tasks are planned for the next release per developer?” or “How many hours are estimated for the next release?” Answers to these questions are of great importance for developers and team leaders alike. The focus on these central questions and the presentation of the answers in the software cockpit (Figure 2) is not perceived as surveillance, but as presenting facts. These facts make the contribution and value of software development transparent and, therefore, constitute a win-win-situation for all parties involved.

(2) Dysfunctional effects of measurement:
Dysfunctional effects of metrics occur in situations in which the act of measurement affects the organization in a counter-productive way, opposite to the intended purpose of the measurement [1, 13]. The reason for measurement dysfunction is that metrics are used as basis for assessment and evaluation although they deviate from the measured objective. Hence, the assessed and evaluated people tend to adjust towards the measured aspects instead to the work objectives. Ironically, an optimization based on measurement data contradicts the accomplishment of the actual objectives: Even though the measured performance seems to improve, true performance declines.

In order to avoid dysfunctional effects, the measurements presented in the software cockpit were used to depict the current status as truthfully as possible. According the approach of “pure informational measurement” [1] no assessments or evaluation were tied to the results. Measurement results are not interpreted as “good” or “bad” and, consequently, visual metaphors such as traffic lights or tachometers with green and red zones were not applied either. Data is presented without any interpretation so that the mere values and collected material is set into focus.

(3) Context-sensitivity of measurement data: The obtained measurement results are, to a great extent, dependent how the software development process is organized and how each individual developer actually contributes to this process. This is why the measurement data presented in the software cockpit can only be interpreted correctly if the interpreter (i.e. the human analyzing the cockpit data) knows about the context of the data. The context is usually best known by people closely related to development. As a result, software cockpits showing context-sensitive data should primarily be designed for developers and closely related roles. The context-sensitivity of the data prohibits its misuse for productivity assessments or as an instrument to set performance goals for development, or else dysfunctional effects would likely take place.

(4) Collecting data of the development process: Unfortunately, important aspects of software development can not be measured automatically. These aspects become measurable only once software development is organized in a way so that meaningful measurable artifacts (e.g., issues reports, time records) are created. However, the creation of such artifacts is mainly a manual activity. Therefore it would lead to an unacceptable overhead if the team leader alone has to collect the data. Hence, developers have to supply the data and, thus, maintain the basis of the software cockpit. For this reason the software cockpit has been designed for team leaders and managers as well as for the individual developers contributing the data. The benefit of feedback from the software cockpit motivates the acceptance of some overhead related to the manual data collection.

(5) Providing information in high quality: High quality information consists of consistent and complete data about which all involved roles share a common understanding. A glossary which precisely defines the semantics lays the foundation of a common understanding.
High quality data is also data that is relevant for the developer. So the developer does not just enter data in order to suffice the prescribed rules, but to actually make use of the data in his or her daily routines. Using one owns data ensures that the data quality remains high and the data is relevant. Low overhead costs, easy data entry and data quality feedback mechanisms contribute to a high quality in the data repositories [18].

(6) Different groups, different data interpretations?
At the involved company, the introduced cockpit encompasses data of several independent development groups that develop for different product parts. Even though the different development groups may develop along a predefined process, documented data and compiled artifacts often have to be interpreted differently compared to other groups. This poses a challenge to the introduction of cockpits that should provide easy-to-understand information across several groups. In particular, problems arise if inter-group data comparisons or aggregations of available data are performed by the cockpit users.

Different inter-group project cultures and, as a result of that, different development processes are the reason for possible different data interpretations. Additionally, different (ly used) development tools among the groups again severe this problem. Before designing complex data extraction algorithms that take account of that, an evaluation whether a consolidation of processes and tools is more reasonable than a fusion of semantically different data sets.

(7) Overzeal in measurement: After first successes to increase transparency in software development, the danger of more and more increasingly complex metrics arises. Since every measure consumes resources for creation, maintenance and evaluation, too many measures bear the risk of high cost and overhead that hampers long-term success. Similarly, complex methods in data mining are usually very costly and do not pay off in the analysis of organizational data. In order to find hidden dependencies in this area, conventional OLAP analysis tools combined with a close contact to the developers is a more promising start than the advanced analysis of imprecise data.

VI. SUMMARY AND LESSONS LEARNED
In this paper we presented our experiences from implementing and introducing a software cockpit at BMD Systemhaus, in a leading Austrian software development company for business software. The introduction followed a defined process with special emphasize on the accompanying process improvement activities and the underlying GQM-based measurement model. The challenges encountered in establishing the software cockpit were mainly socio-psychological in nature and concerned the provision, analysis and presentation of software repository data related to developers.

The following lessons learned summarize our experiences with respect to the introduction of a software cockpit in an industrial setting.

- **Early stakeholder involvement to build trust:** Involve developers from the very beginning in process improvement and the discussions about introduction of the software cockpit. Their involvement and active participation is vital to build the necessary confidence and trust.

- **Unbiased measurement results avoid dysfunctional effects:** Stress the goal to visualize unbiased data to obtain transparency rather than to evaluate employees. Establish purely informational, goal-based measurement. Focus on measuring the output (e.g., implemented requirements) instead of artifacts that are not mission critical (e.g., number of documents or lines of codes).

- **Create awareness of measurement dysfunction:** Discuss measurement dysfunction openly with all involved people, especially management. Clarify that honest engagement is the basis of useful measurement results. ‘Tuning’ measurement results is useless as they do not provide any benefits for developers but distort their overall credibility.

- **Involve management** to sign off the measurement strategy as the basis for the software cockpit. Integrate results from the introduction into institutionalized quality standard documents.

- **Compelling cockpits through high quality data:** Processes that are well understood and produce consistent and correct data lay the foundation for convincing cockpits. Usability and seamless integration of tools for data collection are therefore important.

- **Produce measures useful in everyday activities:** Distinguish between ‘statistical results’ and measures that are anchored in the developers’ everyday activities. Process changes that produce “only metrics” have a high risk of not being successful on the long run and are more prone to bad data quality.

- **Provide immediate feedback about data quality:** An immediate visual feedback about the quality of the data in the cockpit helps to accept some overhead concerning accurate collection of data and improves data quality as developers can cross check existing entries.

- **Care for distinct semantics:** Establish a clear organization-wide terminology where language elements denote what they really stand for. It is important how tools, data fields and processes are named. Language is the mother of thoughts.

- **Simple processes and elementary data lead to understandable measures:** If possible, reduce the complexity of existing workflows, data structures, repositories and data fields during the course of process improvement. Simple processes and elementary data lead to a better understanding of the presented information, allow an accurate interpretation and the anticipation of future trends.

- **Less is more:** Focus your software cockpit on a few simple yet high-quality measures that are frequently consulted, rather than to produce an excessive variety of metrics that no one uses at a regular basis. Cockpit users want to be confronted with just ‘their’ metrics on a single screen.
ACKNOWLEDGMENT

This work has partially been conducted within the competence network Softnet Austria and COMET. We would like to thank the developers and team leaders at BMD for many fruitful discussions that contributed to this paper.

REFERENCES