Concept, Implementation and Evaluation of a Web-based Software Cockpit

Rudolf Ramler\textsuperscript{A}, Wolfgang Beer\textsuperscript{A}, Claus Klammer\textsuperscript{A}, Klaus Wolfmaier\textsuperscript{A}, Stefan Larndorfer\textsuperscript{B}

\textsuperscript{A} Software Competence Center Hagenberg
Sofwarepark 21, A-4232 Hagenberg, Austria
\{rudolf.ramler, wolfgang.beer, claus.klammer, klaus.wolfmaier\}@scch.at

\textsuperscript{B} Software Quality Lab GmbH
Gewerbepark Urfahr 30, A-4041 Linz, Austria
stefan.larndorfer@software-quality-lab.at

Abstract—Software cockpits (software project control centers) provide systematic support for monitoring and controlling the activities in a software development project. Important aspects are to track progress, to visualize team performance, and to provide feedback about the quality of delivered results. Therefore, software cockpits integrate and visualize data from various data sources such as project plans, requirement management, version control, as well as test results. Each of these data sources represents a different perspective on the software project. The integrated view provided by a software cockpit produces a complete and persistent picture of the project status. This paper describes the architecture and functionality of a Web-based software cockpit and its implementation with open source software from the Business Intelligence area. Furthermore, the results and lessons learned from evaluating the practical benefits of the software cockpit in context of a large software development organization are presented.

Keywords—software cockpit; dashboard; business intelligence.

I. INTRODUCTION

The development of software systems is a highly dynamic endeavor that requires continuous monitoring and controlling of the process performance and the quality of delivered products. A software cockpit – also known as software project control center [1] – provides systematic support for monitoring and controlling as it presents accurate information about development activities and critical project states [2]. By integrating, visualizing and exploring project data from different perspectives and at various levels of detail, a software cockpit fosters new insights and reveals the complete picture of what is going on in software development. In case of problems or unexpected deviations, a software cockpit also allows drilling down to present and historical project data to examine and identify the root causes.

The basis of a software cockpit is a measurement infrastructure that periodically extracts data from the different operational data sources of a project, such as project plans, task management, time recording, version control, defect databases as well as the results from daily builds, tests and static analysis. Each of these data sources serves a specific purpose and provides a unique view on the project. For a holistic view, the relevant aspects of these individual views have to be integrated. Therefore, the data is loaded into the cockpit’s central data store, linked along various dimensions, and aggregated to concise metrics and indicators. Depending on the user’s requirements, the data is presented in form of project scorecards, customized reports and interactive dashboards [3]. In addition, the data of the software cockpit provides the basis for several supplementary assessment and prediction approaches, ranging from the simple comparison of plan and actual values to trend analysis, statistical process control as well as reliability and defect prediction models (e.g., [4], [5], [6]).

Today, software cockpits are usually offered as part of comprehensive, commercial tool suites. Thereby, the software cockpit plays the role of an integration point for the data and results produced by the different tools of the tool suite. Typically such integration is restricted to tools from the suite of the same vendor and permits a limited set of predefined usage scenarios. Most projects, however, operate in an established tool and process landscape, which has evolved over years, including a wide range of heterogeneous tools and repositories from different vendors as well as open source software. Moreover, these processes and tools are constantly changing since new technologies, development methods and emerging trends have to be accommodated in order to stay competitive. Thus the applicability of existing cockpit solutions is often limited and, as a consequence, monitoring and controlling cause significant effort due to collecting and processing relevant data. The inherent overhead discourages a full exploration and investigation of the available data about actual costs, work progress and quality.

As an answer to the currently unsatisfying situation in industrial software development, concepts and prototypes for software cockpits have emerged in research and practice. Examples are Specula [7], Hackystat [8], ConQAT [9], or WebME [10]. These software cockpits offer predefined, yet usually customizable solutions for various aspects in measuring as well as evaluating software product quality or process performance. A comprehensive overview of related approaches has been provided in [1]. Project dashboards, which were not included in this overview, are described in [7]. In practice, however, many companies develop their own dashboards [2] designed according to their specific needs and based on spreadsheet or Business Intelligence applications.

This paper focuses on the development of a custom software cockpit. It presents the blueprint of a software cockpit based on concepts from Business Intelligence, which can be implemented with widely available open source tools and frameworks. Business Intelligence, in particular data ware-
housing with its associated ETL (extract, transform, load) technologies (cf. [11], [12]), provides comprehensive support for managerial decision making. Business Intelligence is therefore a rational and widely applicable choice for monitoring and controlling tasks that require collecting, integrating and analyzing software engineering data from heterogeneous tool repositories and project databases [13]. The software cockpit’s visual representation makes use of Web technologies as integration platform to incorporate and align the different functional building blocks into a homogeneous Web-based application (Figure 1). The presented software cockpit has been successfully applied in the context of several industrial projects for monitoring and controlling software development [14]. Hence, the paper is also able to contribute results and lessons learned from the empirical evaluation in a large software development company.

The remainder of the paper is organized as follows. Section 2 gives an overview of the goals and usage scenarios of the software cockpit. Section 3 describes the architecture and implementation of the software cockpit and its different functional building blocks. Section 4 presents the design and results of an empirical evaluation. Section 5 concludes the paper with lessons learned from the software cockpit’s implementation and application.

II. GOALS, SCOPE AND USAGE SCENARIOS OF THE SOFTWARE COCKPIT

A software cockpit supports decision-making in a dynamic, time-driven environment based on up-to-date information. It allows decisions to be based on “hard facts” rather than intuition and personal preferences. Thus, the overall aim in designing a software cockpit is to provide the right information to the right people at the right time. Nevertheless, the design and functionality of a particular software cockpit is determined by the users’ specific needs for information, the way this information will be utilized to control software development, and the availability and quality of the existing data [15].

Thus, in case of the presented software cockpit, the definition of a GQM (Goal-Question-Metric) model consisting of organizational goals, related questions, and required metrics has been a central aspect in the software cockpit’s functional design and the successful introduction to industrial projects [14]. The GQM approach [16] defines a systematic technique to construct goal-centered measurement program. It has been applied in close cooperation with management and team leaders to establish the basis for necessary process improvements and the elicitation of the user requirements. In particular, following goals were addressed with the development of the software cockpit:

G1 Increase the transparency of software development by integrating information fragments from different data sources.

G2 Improve the ability to plan and predict software development activities and potential schedule risks.

G3 Provide timely and accurate information for coordinating development activities across several development teams and project roles.
The identified users and the functionality they require are shown in Figure 2. The users interacting with the software cockpit are: (1) the project manager, who controls the project’s performance, monitors project goals and quality targets, and analyzes the available data to gain additional insights about the project’s status; (2) team members such as developers and testers get feedback about their individual work load and their personal contribution to the overall project performance; (3) management, group leads as well as (4) quality assurance monitor the achievement in terms of project goals as well as quality targets and are able to drill down on details in case of suspicious deviations; (5) external stakeholders are offered customized project status reports; (6) data from tool repositories and project databases is extracted, transformed and loaded into the data warehouse of the cockpit.

![Software Cockpit Diagram](image)

**Figure 2.** Use cases of the software cockpit.

The detailed requirements have been derived from the illustrated usage scenarios. These requirements have been elicited in several cycles of requirements engineering, prototyping, practical application and evaluation. Hence, over time, various solution concepts have been explored in pilot studies and feedback from the software cockpit’s application in different industrial projects has been collected. Still, the graphical as well as the functional design of the software cockpit is under active development. New requirements and usage scenarios emerge with the growing number of users and the cockpit’s application in new contexts. Furthermore, integrating additional open source and third-party components, particularly from the aspiring area of open source Business Intelligence [17], offers new possibilities in presenting and analyzing data, which attracts additional user groups and stimulates further enhancements.

### III. DESIGN AND IMPLEMENTATION

The architecture and design of the software cockpit reflects the specific requirements outlined in the previous section. It is organized in line with the three main tiers depicted in Figure 3. The tiers are realized by a mosaic of open source tools and frameworks that support the software cockpit’s fundamental need for flexibility and extensibility. So, while the overall architecture constitute the stable basis of the software cockpit, individual components such as database adapters, charting libraries or report generators can be selected according to the need of integrating new data sources or additional visualizations and analysis.

A typical Web application consists of multiple tiers that include a data tier, a logic tier, and a presentation tier [18]. The tiers of applications for Business Intelligence cover the data staging area, the presentation services, and the end user access [12]. The architecture of the presented software cockpit (Figure 3) can be divided in three depicted tiers that contain elements of both, Web applications and Business Intelligence. Basic functionality necessary for the operation of the software cockpit, e.g., authentication, authorization, and logging, is embedded within these layers and has been implemented by means of standard technologies.

- **User Interface:** This tier is responsible for an intuitive and flexible communication with the user, providing access to the software cockpit’s functional building blocks such as user defined queries, dashboards and reports. Today’s Web-based user interfaces are typically built upon Web 2.0 technologies. Thus, the resulting Web applications are partly executed on the client, i.e., any Web browser with JavaScript enabled, and show the ability to seamlessly integrate results and interactive elements from different servers and service components within a uniform user interface.

- **Data Analysis:** The software cockpit’s core functionality is to query and analyze the collected data, either based on predefined visualizations and reports or triggered by the user navigating directly through the selected data dimensions. The components implementing the functionality of this tier are executed in an application container on the server.

- **Database and ETL:** The software cockpit relies on the data gathered from the different, heterogeneous and distributed operational data sources. To facilitate reporting and analysis independently from these data sources and the peculiarities of their associated query mechanisms, the data is accumulated in a central data store, i.e. the software cockpit’s data warehouse. The ETL (extract, transform, load) process periodically extracts the relevant data from the operational data sources, transforms the data to optimized storing and retrieval formats, and loads the data into the data warehouse.

![Diagram](image)
Design concepts and details about the implementation of the software cockpit’s functional building blocks are described in the following.

A. Dashboard

The dashboard informs about the project’s progress in terms of incoming, completed and open work items, specifically requirements, change requests and defects. These metrics represent fundamental indicators for planning and controlling software development. The dashboard depicts these metrics graphically on a single page (cf. Figure 1). Therefore, the dashboard has been implemented as Web application that provides a uniform view on the integrated data with different interactive visualization elements, rather than showing separate views on the different data sources side by side.

The visualized data clearly reflects the individual activities in ongoing software development and the charts can easily be interpreted by all team members [3]. Abstract metrics and high-level indicators that are hard to understand have been avoided. To facilitate the exploration of the presented data, mechanisms for in-place analysis allow drilling down from aggregated measurements to individual records. Interactive user interface elements (Figure 4) such as stacked charting along different dimensions, tooltips that provide details about data points, and filters to zoom in on the most recent information have been enabled by Web 2.0 technologies. Furthermore, user specific filters allow personalizing the visualized charts and metrics.

A progress bar shows detailed information about the degree of completion of the scheduled work (Figure 4, right). The progress is calculated from time estimates for resolved and open work items (upper bold bar). The estimates are compared to the actual time spent (lower bold bar). Again, a drill down to the individual entries (lower bar charts) is possible. A grey area in the bar charts indicates data quality issues, e.g., missing estimates, that call for corrective actions.

B. Scorecard

The scorecard defines a list of project metrics that link project management to overall organizational goals and directives. Exemplary metrics are average cycle time per work item or number of un-reviewed changes. While the dashboard primarily addresses the information needs of project managers and team members, the scorecard summarizes the status of project goals for group and quality management.

The definition of metrics including associated target values or thresholds are the center piece of the scorecard. Traffic lights indicate whether a metric meets the target value (green) or deviates beyond defined thresholds (red). Measurement approaches such as GQM support the critical task of deriving appropriate metrics from organizational goals, the accurate definition of these metrics, and the agreement on realistic target values.

The technical realization of the scorecard is straightforward. A Web page containing a hierarchically outlined list of metrics has been sufficient to give an overview of achieved and missed goals at a glance. In case of missed targets, an in-depth analysis of the underlying data allows to investigate and understand the root causes. Hence, for each metric a link to a predefined analysis view – as described in the following – is provided.

C. Analysis

OLAP (On-Line Analytical Processing) supports querying and navigating multi-dimensional data, so-called OLAP cubes, which are stored in a data warehouse. The queries are expressed with MDX (MultiDimensional eXpressions) query language. The results of such queries are usually presented as pivot tables. The software cockpit uses the open source tool JPivot\(^1\) that operates on the open source OLAP server Mondrian\(^2\). JPivot allows the interactive composition of MDX queries via a Web interface that also displays the results in tabular and graphical form.

The integration of the OLAP analysis component extends the software cockpit’s functionality towards flexible ad-hoc queries. Furthermore, predefined queries can also be stored on the server and are linked from other elements of the software cockpit, such as the scorecard.

D. Reports

Reports combine information about a specific subject matter. Usually the audiences for reports are external stakeholders that have to be informed on a regular basis. A typical example is the monthly project progress report provided for customers.

Reporting tools include two main components. First, a report designer is used to create the report templates that describe the data sources, queries and filters for retrieving and preparing the data shown in the report. Furthermore, it specifies the

---

1 http://jpivot.sourceforge.net
2 http://mondrian.pentaho.org
layout and formatting of the report and its elements. Second, a report generator is used to produce the actual report documents based on the description and specification in the report template. It supports different output formats such as HTML or PDF for use with standard document viewers.

The software cockpit uses the open source reporting system Business Intelligence and Reporting Tools (BIRT)\(^3\) to provide reporting functionality. Thus, the BIRT report generator has been set up as part of the software cockpit to produce reports from an extensible list of predefined report templates stored on the server. The reports are created on the fly, once they are opened in the Web browser.

E. Data Warehouse

The data warehouse contains the entire data that can be queried and analyzed as part of the software cockpit. Its main purpose is the integration of data from heterogeneous, distributed operational data sources. The data in each of the operational systems represents a different view on a software project. By combining these views, a complete and persistent picture of the project status can be produced. Yet, the data warehouse fulfills two additional, elementary requirements. First, it ensures independence from operational systems and their specific data formats and interfaces, which are prone to change over time, when development processes are revised or new tools are introduced. Second, it distills a consistent and reliable “single version of truth” [11] from the various data fragments kept in the different data sources.

The data in the warehouse is organized in a star schema [12] for easy querying and navigating the data along different dimensions and hierarchies. The open source OLAP server Mondrian is used to manage the multi-dimensional data, which is stored in the open source database management system MySQL\(^4\).

F. ETL Process

Several times a day, the data warehouse is updated with actual data from the operational systems to provide a timely overview of the progress in software development. The automated integration of the data from the disparate tool repositories and project databases with varying levels of data quality and inconsistent semantics is a complex and labor-intensive task [19]. ETL technologies support the extraction, transformation and loading of the data from disparate sources. Besides individual import scripts, the open source tool Kettle\(^5\) has been applied to automate the ETL process. Kettle allows graphical modeling of the ETL process and provides numerous adaptors to attach to a wide range of data sources, which includes relational databases, XML files, most common document formats, and even RSS feeds.

---

\(^{3}\) [http://www.eclipse.org/birt](http://www.eclipse.org/birt)

\(^{4}\) [http://www.mysql.com](http://www.mysql.com)

\(^{5}\) [http://kettle.pentaho.org](http://kettle.pentaho.org)
IV. Results from the Evaluation

Software cockpits are considered a valuable instrument for monitoring and controlling software development. Unfortunately, there is little empirical evidence – exceptions are [20] and [21] – that endorses the benefits of software cockpits in an industrial environment. This section presents the evaluation results from application of the described software cockpit in a large software development organization.

A. Study Design

Ciolkowski et al. [20] introduced a design for the evaluation of software cockpits in industrial case studies. This study design has been adapted and enhanced for the evaluation of the software cockpit presented in this paper. It involved the following aspects on which information was gathered from the users of the software cockpit.

- **Overall goal accomplishment.** As described in Section 2, the development of the software cockpit had three main goals: (G1) Increase transparency of software development, (G2) improve the ability to plan and predict, and (G3) support coordination of development activities. The fulfillment of these initial goals was addressed in the questionnaire by a five-point Likert scale (“agree”, “rather agree”, “neutral”, “rather disagree”, and “disagree”).

- **Usefulness and ease of use.** The evaluation of these factors is based on the Technology Acceptance Model (TAM) [22] that models the decision about how and when users will adopt a new technology: (1) Perceived usefulness, defined as “the degree to which a person believes that using a particular system would enhance his or her job performance” [23]. (2) Perceived ease-of-use, defined as “the degree to which a person believes that using a particular system would be free from effort” [23].

- **Improvement potential.** Comments on positive aspects and suggestions on how to improve the software cockpit were collected with a set of open questions.

B. Execution

The software cockpit was applied in a large software development organization concerned with the development and maintenance of financial applications. The evaluation was part of an anonymous questionnaire used to appraise the results from the process improvement project, in which the software cockpit was introduced. The questionnaire was sent to about 240 persons from the development and the support department. 92 questionnaires were returned. The participants from the development department (53 persons) were directly affected by the software cockpit and, thus, selected for the evaluation.

Following figures briefly characterize the involved participants. 76 percent of the study participants where developers and engineers; the remaining 24 percent were project managers (11 percent) or group leads (13 percent). Close involvement of management is indispensable for the successful application a software cockpit and, furthermore, management represents one of the main user groups. Quality management has been involved in the development of the software cockpit and the preparation of the study and was therefore excluded from the evaluation.

Study participants were also grouped according their experience they had in their role. 59 percent of participants were recognized as very experienced (i.e., more than 3 years of experience), 32 percent as experienced (i.e., 1 to 3 years), and 9 percent of the participants had little experience (i.e., less than 1 year). These results strengthen the confidence in the participants’ competence to provide qualified answers.

C. Results from Evaluating Overall Goal Accomplishment

For each of the three goals of the software cockpit (cf. Section 2), the study participants expressed their agreement to what extent the goal has been accomplished.

Figure 5 shows the evaluation results for the first goal G1: *Increase the transparency of software development by integrating information fragments from different data sources*. This goal received the strongest acceptance among all three goals. 86 percent of the study participants evaluated the goal accomplishment positive (i.e., they were able to “agree” or “rather agree” that this goal has been accomplished), 14 percent evaluated the goal accomplishment “neutral”, and no one evaluated the goal accomplishment negative (i.e., had to “disagree” or “rather disagree”).

![Figure 5. Accomplishment of the goal to increase transparency.](image)

Figure 6 shows the evaluation results for the second goal G2: *Increase the ability to plan and predict software development activities and potential schedule risks*. As depicted, the accomplishment of this goal has been evaluated positive by the study participants. 76 percent “agreed” or “rather agreed”, 14 percent were “neutral”, and 10 percent “disagreed” or “rather disagreed”.

![Figure 6. Accomplishment of the goal to improve planning and prediction.](image)
Figure 7 shows the evaluation results for the third goal G3: Provide timely and accurate information for coordinating development activities across several development teams and project roles. This goal received a neutral to slightly positive evaluation. 32 percent of the study participants “agreed” or “rather agreed”, 51 percent were “neutral”, and 17 percent “disagreed” or “rather disagreed” that this goal had been accomplished. This goal was found the hardest to achieve as the coordination of teams required additional changes to the established processes.

Figure 7. Accomplishment of the goal to support coordination.

In total, the accomplishment of two of the three goals was evaluated positive; the accomplishment of the third goal was evaluated neutral to slightly positive. In terms of accomplished goals, the development and introduction of the software cockpit can therefore be considered successful.

D. Results from Evaluating Usefulness and Ease of Use

For the two factors, the study participants expressed their agreement to what extent they perceived the software cockpit as useful and easy of use in context of their work.

Figure 8 shows the evaluation results for the factor Usefulness. This factor has been evaluated “neutral” to slightly positive. 35 percent of the study participants “agreed” or “rather agreed”, 50 percent were “neutral”, and 15 percent “disagreed” or “rather disagreed”. A detailed analysis of the results showed, that many developers answered neutral, whereas the majority of the participants involved in management perceived the software cockpit as useful. As confirmed by the analysis of the optional suggestions for improvement, participants that disagreed with the usefulness of the software cockpit often required further information or additional functionality.

With a neutral to slight positive evaluation for the factor usefulness and a positive evaluation for the factor ease of use, the acceptance of the software cockpit can therefore be considered successful, albeit some open issues with a negative impact on the perceived usefulness were identified.

E. Results from Evaluating the Improvement Potential

Via open questions the participants were able to provide positive feedback (14 comments) and suggestions for improvement (10 comments). The comments were examined and set in relation to the evaluation results about goal accomplishment as well as perceived usefulness and ease of use. Positive comments were mostly linked to the increased transparency in software development. Improvement suggestions indicated the need for additional functionality, e.g., extra filter settings, or a lack of awareness about the potential benefits provided by the software cockpit, which explained the “disagreement” by some of the participants.

V. SUMMARY AND LESSONS LEARNED

This paper presented the design and architecture of a Web-based software cockpit drawing from established concepts of Business Intelligence, in particular, ETL processes, data warehousing, reporting, and OLAP. The software cockpit has been implemented with a mosaic of open source tools and frameworks, which were assembled to a coherent Web application. Furthermore, the software cockpit was successfully applied in different industrial projects and has been evaluated in the context of a large software development organization. The evaluation results confirm the general applicability of the proposed approach and demonstrate that the software cockpit has been able to achieve its initial goals, which were increase transparency in software development (evaluated positive), improve the ability to plan and predict (evaluated positive), and support coordination of development activities (evaluated slightly positive).

Throughout the design and implementation of the software cockpit as well as from its application and evaluation, a number of lessons learned have been distilled, which will give direction to our future work.
A considerable part of the software cockpit’s functionality is constituted by widely available open source tools and frameworks. Integrating these third party components provides several practical benefits, such as low risk as many of the components are well-established tools and frameworks, low upfront costs due to the fact that mostly open source software has been used, and the ability to rapidly deliver new functionality. In future, we consider constructing the software cockpit as mashup [24] to facilitate convenient integration of further functionality and data.

Successful development of a software cockpit is not (only) a question of implementing the required functionality, but primarily a question of finding relevant metrics and suitable indicators for monitoring and controlling software development activities. We applied the GQM approach for defining the underlying measurement strategy and used a goal-oriented requirements engineering approach for designing the data warehouse. In future, we aim at the employment of GQM+Strategies [25] to explicitly link measurement to organizational goals.

In order to benefit from a software cockpit, an organization requires established development processes, understanding of cause-effect relationships of relevant indicators, and sufficient sustainability of measurement programs. Thus, in many cases, the development of a software cockpit has to be accompanied by process improvement and standardization initiatives. Part of our future work will be the identification of best practices in software development that allow taking full advantage of the software cockpit’s functionality.

The majority of the effort spent on the development and introduction of a software cockpit is consumed by data integration. Automating the process by which data is extracted from operational data sources, transformed, integrated and loaded into the data warehouse is technically challenging. Data quality issues hamper the implementation of a repeatable ETL process and, furthermore, limit the usefulness and reliability of the software cockpit. In future, we plan to expand the software cockpit’s visual capabilities to provide immediate feedback in case of poor data quality including semi-automated correction mechanisms.

REFERENCES