Business Intelligence in Software Quality Monitoring: Experiences and Lessons Learnt from an Industrial Case Study

Alexander Kalchauer[2], Sandra Lang[1], Bernhard Peischl[1], and Vanesa Rodela Torrents[2]

[1] Softnet Austria, Inffeldgasse 16b/II, 8010 Graz, Austria
{bernhard.peischl,sandra.lang}@soft-net.at
http://www.soft-net.at
[2] Technische Universität Graz, Institut für Softwaretechnologie, Inffeldgasse 16b/II, 8010 Graz, Austria
{kalchauer,torrents}@student.tuGraz.at

Abstract. In this industrial experience article we briefly motivate continuous, tool-supported monitoring of quality indicators to facilitate decision making across the software life cycle. We carried out an industrial case study making use of a web-enabled, OLAP-based dashboard to measure and analyse quality indicators across various dimensions. We report on lessons learnt and present empirical results on execution times regarding different groups of queries implementing the desired quality metrics. We conclude that today's business intelligence (BI) solutions can be employed for continuous monitoring of software quality in practice. Furthermore, BI solutions can act as an enabler for any kind of data-driven research activities within companies. Finally we point out some issues, that need to be addressed by future data-driven research in the area of software measurement.

Keywords: business intelligence, lessons learned, industrial case study, web-enabled software dashboard.

1 Motivation

The new generation of interconnected and open IT systems in areas like healthcare, telematics services for traffic monitoring, or energy management are evolving dynamically, for example, software updates are carried out in rather short cycles under presence of a complex technology stack. Tomorrow's IT systems are liable to stringent quality requirements (e.g. safety-, security or compliance issues) and evolve continuously, that is, the software habitually is subject of change. Given this context, monitoring of quality attributes becomes a necessity. Efficient and effective management of a software product or service thus requires continous, tool-supported monitoring of well-defined quality attributes.

The role of software as a driver of innovation in business and society leads to an increasing industrialization of the entire software life cycle. Thus, the development and operation of software nowadays is perceived as mature engineering

D. Winkler, S. Biffl, and J. Bergsmann (Eds.): SWQD 2014, LNBIP 166, pp. 34–47, 2014.

discipline (even in the secondary sector) and the division of labor is driven by coordinated tools (in the sense of an integrated tool chain). As a consequence, monitoring of quality attributes has to be carried out holistically, i.e. taking into account various quality dimensions. This includes the continuous monitoring of process- (e.g. processing time of for bugs or change requests), resource- (e.g. alignment of bugs to software engineers) and product-metrics (e.g. code metrics like test coverage as well as metrics regarding abstract models of the software, if in place). However, to establish a holistic view on software quality we lack methods and tools that allow one for an integration of the relevant dimensions and (key) quality indicators.

To address these issues, as part of the research programme Softnet Austria II, a web-enabled software dashboard making use of powerful OLAP technology has been developed. This cockpit strives to provide the technological underpinning to enable the integration of the various dimensions. In this article we briefly describe the software architecture of our dashboard and point out experiences and lessons learnt in the course of a pilot project. We considered issue management, that is the treatment of defects and change requests (CRQs) alongside with related parts regarding project management. In this industrial experience paper we present exhaustive empirical results regarding the running time of the queries implementing the various quality indicators. We contribute to the state of the art in the field of quality monitoring in terms of (1) providing experiences on setting up a software dashboard relying on an open-source BI solution and (2) a performance analysis regarding the execution times.

In Section 2 we discuss the importance of metrics and quality models and point out the need for integration of the different views on software quality. In Section 3 we subsume the main ideas behind OLAP, typical usage scenarios and operations and the overall architecture of our web-enabled dashboard solution. In Section 4 we present our industrial case study conducted with a company from the secondary sector[1]. In the context of the prevailing processes around CRQs we outline (1) examples of quality metrics, (2) show how to use the OLAP technology for ad-hoc analysis of an industrial software repository, and (3) provide empirical insights on the execution time of the various mulit-dimensional queries. In Section 5 we report on lessons learned from our industrial case study, Section 6 discusses related work and Section 7 points out open issues and concludes our industrial experience paper.

2 Measuring Software Quality

In general software quality is nowadays perceived as the degree of fulfilment of either explicitly specified or tacit requirements of the stakeholder (e.g. the ISO/IEC 25010:2011 standard [1] follows this notion). A specific quality model makes this abstract definition applicable. Usually a quality model defines certain quality attributes (e.g. test coverage on the level of code, test coverage on

[1] Due to a non-disclosure and confidentality agreements we do not mention our cooperation partner and made relevant data anonymous.

the level of requirements, processing time of defects or CRQs, testability of the product, ...) and allows one for refining these attributes in terms of specific, measurable characteristics of the resources, the product or the underlying processes. The quality model thereby specifies a hierarchy of measurable characteristics and ideally allows one for definition, assessment and prediction of software quality [7]. Although definition, assessment and prediction of quality are different purposes, the underlying tasks are not independent of each other. Obviously, it is hard to assess quality without knowing what it actually constitutes. Likewise one cannot predict quality without knowing how to assess it [7]. Quality models may serve as a central repository of information regarding software quality. Ideally the different tasks in software quality engineering, e.g. definition, assessment, and prediction of quality attributes should rely on the same model. However, in practice different (often implicit and incomplete) models are used for these tasks. Therefore a common infrastructure that supports definition, assessment and prediction of software quality using a common representation of the underlying quality-relevant data and metrics is highly desireable. As mentioned previously, our dashboard solution focuses on continuous monitoring of the software life cycle particularly addressing the need for a smooth integration of various quality dimensions. This allows one for analysing the relevant aspects regarding quality attributes, including the early detection of erroneous trends. The early detection of trends in turn is the foundation of setting remedial countermeasure in motion.

3 Exploiting Business Intelligence in Software Engineering

The main idea behind BI is to transform simple data into useful information which enhances the decision making process by basing it on a wide knowledge about itself and the environment. That minimizes risks and uncertainties derived from any decision that a company has to take [17].

Moreover, business intelligence contributes to translate the defined objectives of a company into indicators with the possibility of analysing them from different points of view. That transforms the data into information that, not only is able to answer questions about current or past events but it also makes possible building models to predict future events [6].

3.1 Architecture of the Web-Enabled Dashboard

The concepts outline in the previous sections are typically used in various business areas for measurement, analysis and mining of specific data pools. Our dashboard primary serves the purpose of carrying out data-driven research particularly addressing the integration of various views on software quality (process-, product- and resources view). Figure 1 outlines the architecture of the web-enabled dashboard making use of BI technology. The dashboard uses the open source tool JPivot [10] as front end. It operates on the open source OLAP

server Mondrian [4]. JPivot allows the interactive composition of MDX (Muli-Dimensional eXpressions) queries via a Web interface that also displays the results in tabular and graphical form. Predefined queries are stored on the server and accessible from the web interface. JPivot loads these queries from the query files or takes them from the MDX editor provided through the web interface, than OLAP parses those queries through the multidimensional cubes and converts them to SQL queries which are sent to a MySQL database.

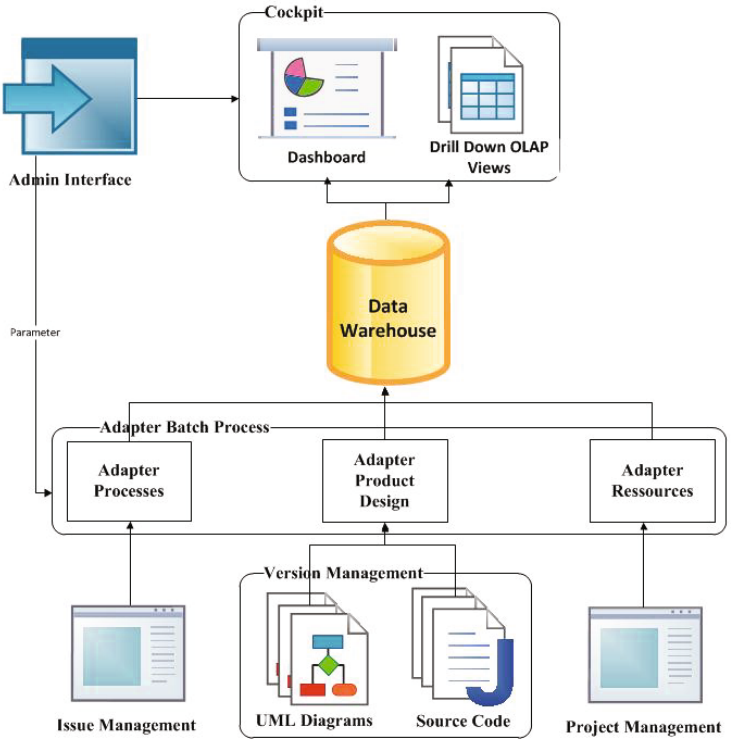

Fig. 1. Software architecture of the BI-based dashboard [11]

4 Industrial Case Study: Quality Indicators for Managing Defects and Change Requests

Our cooperation partner is developing a software product and has well-defined work flows for development and maintenance in place. For our pilot project we addressed the work flows around CRQs. Our cooperation partner classifies issues according to three types: Defects, Enhancements (CRQs) and Tickets. In general issues pass through various phases: The decision phase (Decide), the implementation phase (Implement), the assessment phase (Review) and the test phase (Test). According to these categorization, the work flow around CRQs is partitioned into these four areas as illustrated in Figure 2.

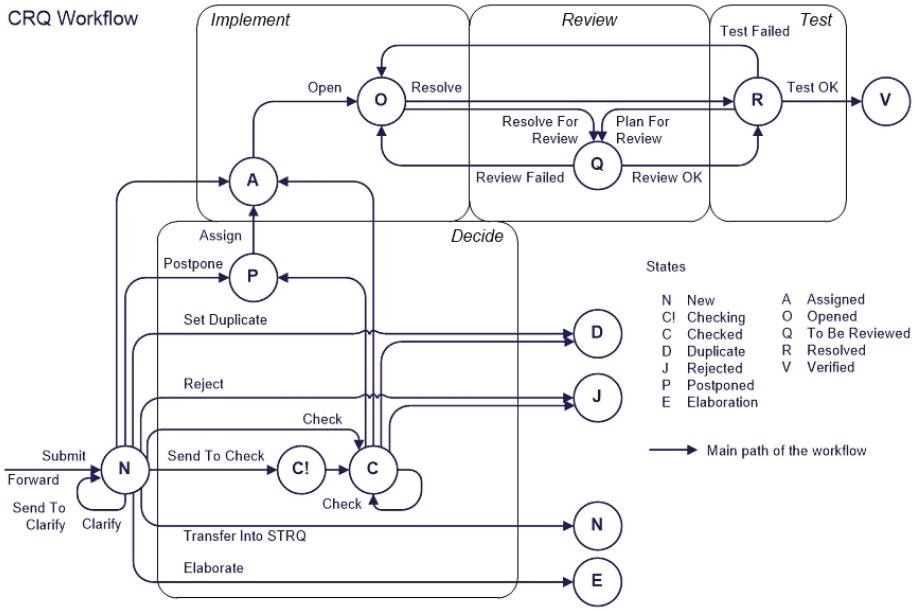

Fig. 2. Simplified workflow, states and state transitions

4.1 Measuring the Management of Defects and Change Requests

Regarding the monitoring of quality indicators we distinguish between metrics with a target state from the set of inflow states {A, C, C!, D, E, N, J, P} and metrics with target state from the set of outflow states {O, Q, R, V}. Relying on this categorization, metrics have been grouped into 6 groups:

- **Group 1:** Metrics regarding the inflow states and the elapsed time from submission of an issue to a particular state. The around 20 performance indicators from this group affect the decision phase and the implementation phase. Depending on the concrete metric, drill-down and slicing into 15 to 20 different dimensions (version, milestone, product, status, priority, error severity, person, role, age etc.) is required.
- **Group 2:** Metrics regarding the outflow state and the elapsed time from submission to a particular state. The around 15 performance indicators concern the review- and testing phase and have similar requirements wrt. ad hoc analysis as group 1.
- **Group 3:** Metrics regarding the inflow states and the elapsed time on that state. The 11 performance indicators are dealing with the elapsed time of a CRQ in the decision- and implementation phase. Some of the performance indicators are drilled-down to around 15 dimensions, often standard aggregates (maximum, minimum, average) are used.

- **Group 4:** Metrics regarding outflow states and the elapsed time on that state. The 14 performance indicators require ad-hoc analysis in up to 20 dimensions.
- **Group 5 and Group 6:** These metrics have high level of complexity as these metrics relate states that are not consecutive. The complexity is caused due to the fact, that it is necessary to keep track of an issue from its submission through all its changes in order to compute the desired metrics. Nevertheless, these performance indicators need to be drilled down to a couple of dimensions. However, these metrics are less time critical as the Group 1-4 as the quality indicators regarding group 5 and 6 are used occasionally only.

4.2 Queries for Ad-Hoc Analysis

The *MDX* language has become the standard defined by Microsoft to query OLAP systems and provide a specialized syntax for querying and manipulating the multidimensional data stored in OLAP cubes.

Table 1 shows the example of a query and the explanation of every statement following by its graphical representation. As one can see in the query four dimensions are selected:

- Time and measures on the columns.
- Products and severities on the rows.

Figure 3 and 4 illustrates the basic idea.

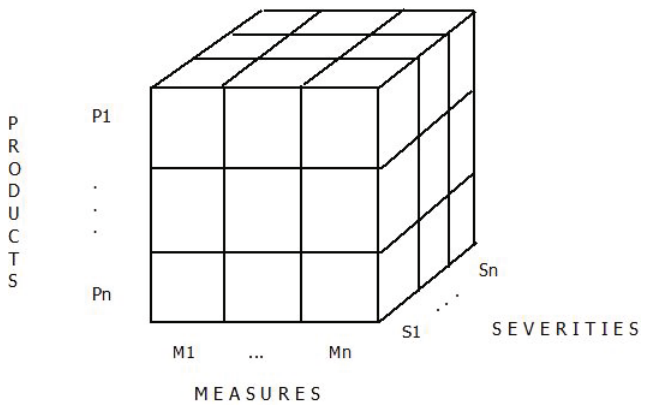

Fig. 3. The three queried dimensions form a cube

Using the *where* clause, one has the possibility to slice the represented cube. That means that not all the data of the four dimensions will be shown, but only those specified through the slicer defined in the *where* clause. In this case, these issues which go to the state *Resolved (R)* and were successfully tested in the previous step (*Verified (V)*).

Table 1. Query Example

	Functions	*Handled data*	*Description*
1	Select NON EMPTY Crossjoin	([LastChangedDate].[All date].children, [Measures].[IssuesStatesCount], [Measures].[MinAgeInDays], [Measures].[MaxAgeInDays], [Measures].[AvgAgeInDays])	The crossjoin function returns the cross product of two sets. It is necessary because LastChangedDate and Measures are two different dimensions. Since the type given to crossjoin has to be a set, one converts the members to a set by enclosing them in curly brackets.
2	ON COLUMNS,		All the information selected before will be shown through the x axis.
3	NON EMPTY Crossjoin	([Product].[All product].Children, [Severity].[All severity])	
4	ON ROWS		All the information selected before will be shown through the y axis.
5	From [SoftCockpit]		Cube from where one wants to extract the data
6	where	([PrevStatus].[All status].[Verified], [Activity].[All activities].[resolved])	Slicer: Tuple which contains the members that specify the conditions that the data has to fit to be shown.

The output given by the application is a table showing the required information on the desired rows (Figure 5). Products and Milestone can be drilled down, since they compose a hierarchy with several levels.

4.3 Performance Analysis

During the implementation of the different sets of quality indicators, differences in the performance regarding the amount of data have been observed. To analyse this, different sets of data of the fact table have been taken into account. The following graphs and explanations show the conclusions obtained from this performance analysis. Table 2 summarizes the ressources of the server on which we conducted our experiments. The test has been executed as follows: Every developed query has been tested for a different amount of years and consequently for a different amount of data.

Moreover, a relation between some queries has been noticed during the experiment as queries belonging to the same group have similar execution time

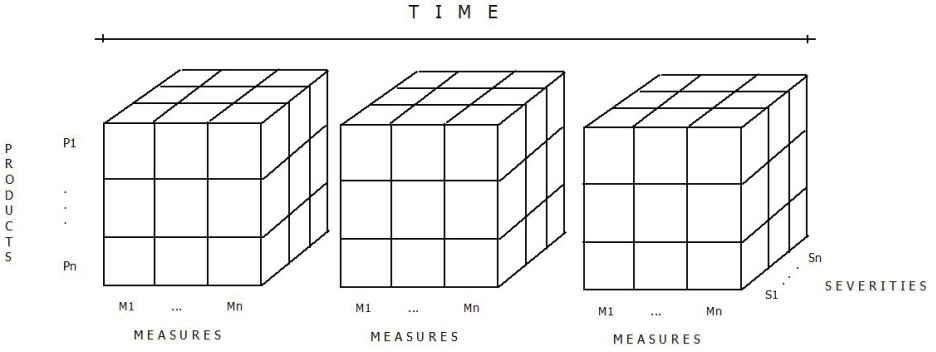

Fig. 4. The three previous dimensions are queried along the years which is the fourth dimension

		LastChangedDate							
		2011				2012			
		Kennzahlen				Kennzahlen			
Product	Milestone	IssuesStatesCount	MinAgeInDays	MaxAgeInDays	AvgAgeInDays	IssuesStatesCount	MinAgeInDays	MaxAgeInDays	AvgAgeInDays
Product A	All milestone	8	25,04	1.008	291,77	2	13	205	109
Product B	All milestone					4	20	73	33,25
Product C	All milestone	1	1	1	1				
Product D	All milestone	1	28	28	28	1	32	32	32
Product E	All milestone					1	94,96	94,96	94,96

Fig. 5. Results produced by the query of Table 1

Table 2. Used resources

Physical Memory: 32GB
JVM Memory: 2GB
Processor: Intel Xeon CPU ES606@ 2.13 GHz, 2.13GHz (2 Processors)
OS: Windows Server 2008 R2 Standard

characteristics. The reason why this happens can be found in the complexity of the queries which is caused by the where clause (slicer) of the query.

For the different groups of queries the following execution times have been collected:

Figure 6 shows the execution times for the queries of Group 1 with different sets of data. The average response time of all the considered metrics of Group 1 is shown in Figure 7.

Figure 8 shows the execution times for the queries of Group 2 with different sets of data. Queries 2.7 and 2.8 are excluded since they proof the correct use of the open state and have been implemented for the purpose of verification. These queries are not used for continuous monitoring of the process. To ease the illustration, queries 2.10 and 2.13 are also excluded as they can be computed

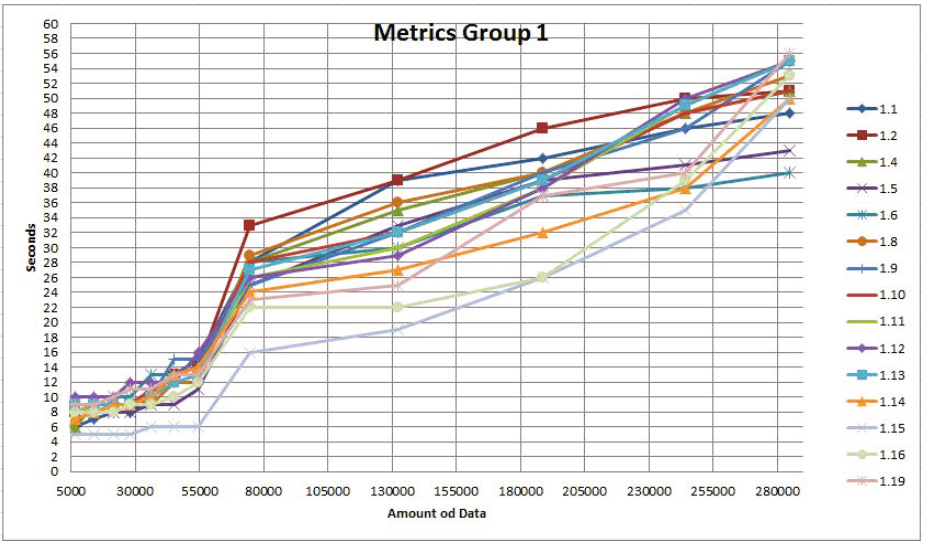

Fig. 6. Response time regarding the amount of the data for Group 1

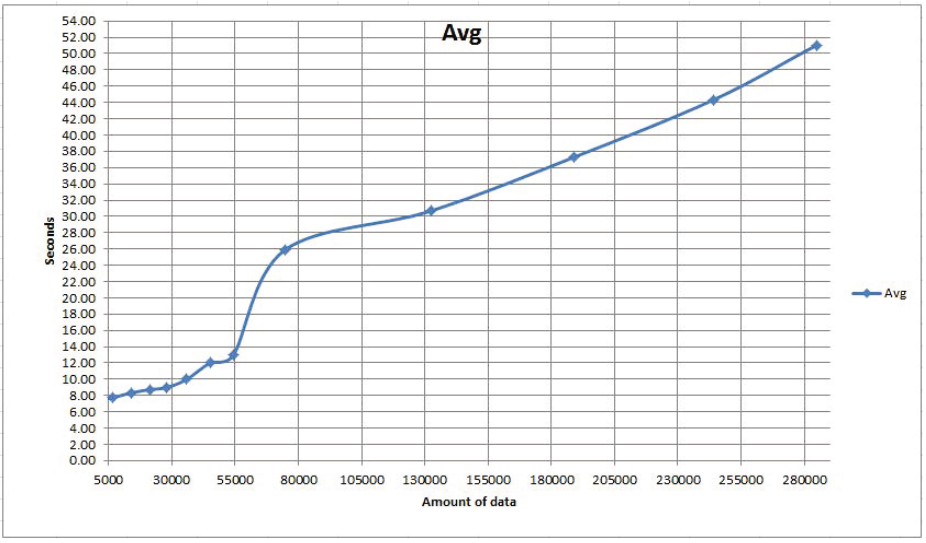

Fig. 7. Average over the response time of Group 1

within a second even for a big amount of data.The average of all the quality indicators of Group 2 is shown in Figure 9.

Execution times for the queries of Group 3 and 4 look very similar. For Group 5 and Group 6 queries we did not evaluate the response times, because of the complexity of these queries. However, as mentioned before, these queries are not used for continuous quality monitoring and are executed only occasionally.

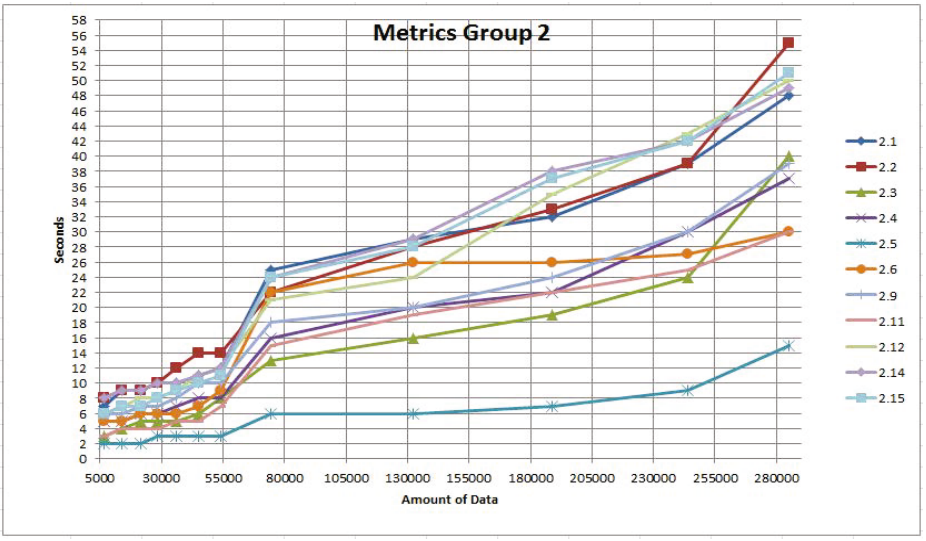

Fig. 8. Response time regarding the amount of the data for Group 2

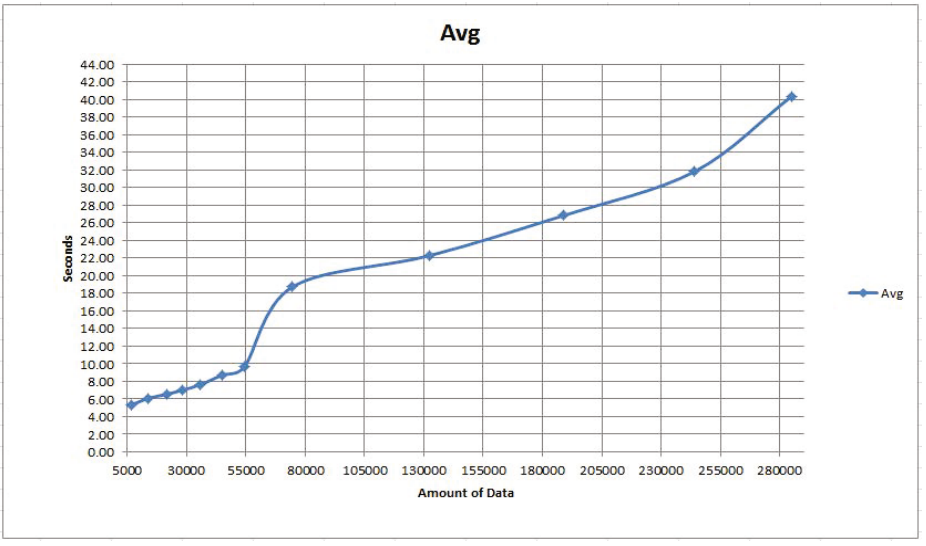

Fig. 9. Average over the response time of Group 2

However, inside these two groups one can also differentiate between the metrics which start in state New ('N', creation of an issue) and go to a non-terminating state (we refer to a middle state in the following, see Figure 2) and the metrics which start in a middle state and end in another middle state. Since the way through the hierarchy is longer for those metrics starting in state *New* ('N'), the complexity of these metrics is higher. Furthermore the number of OR

operations in their definition is higher, which increases the execution times further. Therefore a comparison between metrics from state new ('N') to a middle state and the metrics which goes from a middle state to another middle state can be of interest.

Figure 10 shows the differences in performance between those metrics.

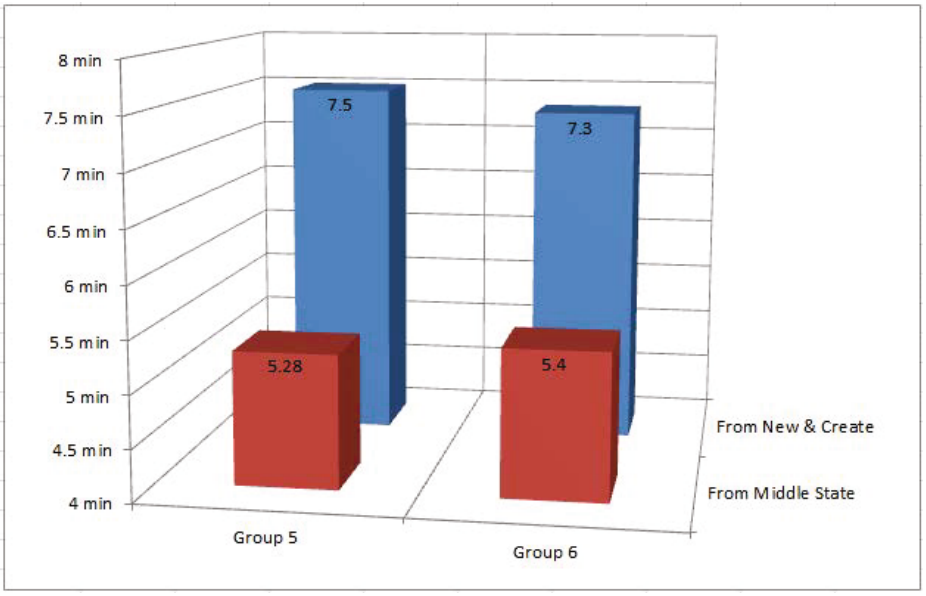

Fig. 10. Performance comparison between initial and middle states queries

5 Discussion and Lessons Learned

As one can observe, in all the metrics groups (although the execution times are different) the shape of the graph is the same. The execution time increases in a mostly linear way up to approx. 55000 rows in the fact table. For bigger fact tables, there is a dramatic increase in the running time which is shown by the almost vertical line. From this point, the execution time is increasing following an almost linear shape. This fact shows that, with the current resources, if the amount of data to be queried is above 55000 rows for the fact table, a generalized tuning process should be carried out.

For very large cubes, the performance issues are driven mostly by the hardware, operating system and database tuning, than anything Mondrian can do.

It has been observed that for this amount of data, increasing memory resources means no change, since the execution times keep on being the same. 2GB are enough to carry out any operation with this data. Therefore a tuning for the database and Mondrian is suggested in [19].

6 Related Work

Software cockpits for interpretation and visualization of data were already conceptually prepared ten years ago. A reference model for concepts and definitions around software cockpits is presented in [15]. Concepts and research prototypes have been developed in research projects like Soft-pit [14] and Q-Bench [9]. Whereas Soft-pit has the goal of monitoring process metrics [14], the Q-Bench project is aimed at developing and using an approach to assure the internal quality of object oriented software systems by detecting quality defects in the code and adjusting them automatically.

The authors of [2] report that the integration of data from different analysis tools gives a comprehensive view of the quality of a project. As one of the few publications in this field, the authors also deal with the possible impact of introducing a software dashboard. The authors of [3], [5], [13], [12] report on experiences and lessons learnt regarding software dashboards .

According to [8], in the context of explicit quality models, a viable quality model has to be adaptable and the handling of the obtained values (i.e. a table result) should be easily understandable. Business intelligence systems with OLAP functionalities support these requirements and are used nowadays in several fields [20]. Thefore using BI technology for qualiy monitoring offers an excellent foundation for data-driven research in collaboration with companies [16].

Related to the work presented herein are industrial-strength tools, e.g. Bugzilla (http://www.bugzilla.org), JIRA (http://www.atlassian.com/software/jira), Polarion (http://www.polarion.com) and Codebeamer (http://www.intland.com), Swat4j (http://www.codeswat.com), Rational Logiscope (http://www-01.ibm.com/software/awdtools/logiscope/) or Sonar (http://www.sonarsource.org) strive to integrate quality metrics in an environment of heterogeneous data sources [18]. Some tools (Swat4j or Rational Logiscope) support the evaluation of quality metrics explicitly taking account a quality model. Whereas these tools are very flexible in data storing and representation, they offer few possibilities for the integration and analysis of different quality dimensions.

7 Conclusion

In this article we motivate that continuous, tool-supported quality monitoring is gaining more and more importance due to a new generation of interconnected and open IT systems that evolve dynamically under presence of a complex technology stack. We briefly summarize the challenges in operationalising quality indicators and point out the necessity to integrate the different quality perspectives (e.g. process-entered view, product-centred view and resource-centred view). Having a multi-dimensional view in mind, we argue that OLAP technology, as it is successfully applied in various business areas - provides a solid foundation for data-driven research in the field of software quality. We provide

an overview of the software architecture of our software dashboard which makes use of open-source OLAP technology. Afterwards we present an industrial case study carried out with a company developing a software product. The case study particularly deals with quality indicators for managing issues (change requests and defects) and comprises over 43.000 issues over a time period of 13 years. We implemented over 100 metrics that are dissolved in up to 20 different dimensions. Our exhaustive analysis of the running time for the most important quality indicators shows that today's BI technology is good enough to support ad-hoc analysis for most of the relevant quality indicators. Further we report on challenges and lessons learnt. The work presented in this industrial experience paper shows that - by relying on today's BI technology - measurements and ad-hoc analysis of relevant data is feasible, however, there is an increased need for further research dealing with the impact of measurement tools in industrial practice. Yet it is an open issue, how continuous monitoring of quality indicators effects the overall software quality. Further research should take advantage of past experiences and lessons learnt but specifically address the impact of continuous quality monitoring on software quality.

Acknowledgement. The work presented herein has been partially carried out within the competence network Softnet Austria II (www.soft-net.at, COMET K-Projekt) and funded by the Austrian Federal Ministry of Economy, Family and Youth (bmwfj), the province of Styria, the Steirische Wirtschaftsfrderungsgesellschaft mbH. (SFG), and the city of Vienna in support of the Center for Innovation and Technology (ZIT). We listed the authors in alphabetical order.

References

1. ISO/IEC 25010:2011 software engineering - software product qualtiy requirements and evaluation (SQuaRE) - quality model. International Organization for Standardization (2011)
2. Bennicke, M., Steinbrückner, F., Radicke, M., Richter, J.-P.: Das sd&m software cockpit: Architektur und erfahrungen. In: INFORMATIK, pp. 254–260 (2007)
3. Biehl, J.T., Czerwinski, M., Smith, G., Robertson, G.G.: Fastdash: a visual dashboard for fostering awareness in software teams. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 1313–1322. ACM (2007)
4. Bouman, R., van Dongen, J.: Pentaho Solutions: Business Intelligence and Data Warehousing with Pentaho and MySQL. Wiley Publishing (2009)
5. Ciolkowski, M., Heidrich, J., Simon, F., Radicke, M.: Empirical results from using custom-made software project control centers in industrial environments. In: Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 243–252. ACM (2008)
6. Dario, B.R.: DATA WAREHOUSING: Investigacin y Sistematizacin de Conceptos. PhD thesis, Universidad nacional de Crdoba (2009)
7. Deissenboeck, F., Juergens, E., Lochmann, K., Wagner, S.: Software quality models: Purposes, usage scenarios and requirements. In: ICSE Workshop on Software Quality, WOSQ 2009, pp. 9–14 (2009)

8. Deissenboeck, F., Juergens, E., Lochmann, K., Wagner, S.: Software quality models: Purposes, usage scenarios and requirements. In: ICSE Workshop on Software Quality, WOSQ 2009, pp. 9–14. IEEE (2009)
9. http://www.qbench.de/
10. JPivot, http://jpivot.sourceforge.net/
11. Lang, S.M., Peischl, B.: Nachhaltiges software management durch lebenszyklusübergreifende überwachung von qualitätskennzahlen. In: Tagungsband der Fachtagung Software Management, Nachhaltiges Software Management. Deutsche Gesellschaft für Informatik (November 2012)
12. Larndorfer, S., Ramler, R., Buchwiser: Dashboards, cockpits und projekt-leitstände: Herausforderung messsysteme für die softwareentwicklung. OBJEKTspektrum 4, 72–77 (2009)
13. Larndorfer, S., Ramler, R., Buchwiser, C.: Experiences and results from establishing a software cockpit at bmd systemhaus. In: 35th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2009, pp. 188–194 (2009)
14. Münch, J., Heidirch, J., Simon, F., Lewerentz, C., Siegmund, B., Bloch, R., Kurpicz, B., Dehn, M.: Soft-pit - ganzheitliche projekt-leitstände zur ingenieurmässigen software-projektdurchführung. In: Proceedings of the Status Conference of the German Research Program Software Engineering, vol. 70 (2006)
15. Münch, J., Heidrich, J.: Software project control centers: concepts and approaches. Journal of Systems and Software 70(1), 3–19 (2004)
16. Peischl, B., Lang, S.M.: What can we learn from in-process metrics on issue management? Testing: Academic and Industrial Conferencem Practice and Research Techniques, page IEEE Digial Library (2013)
17. Raisinghani, M.S.: Business intelligence in the digital economy: opportunities, limitations and risks. Idea Group Pub. (2004)
18. Staron, M., Meding, W., Nilsson, C.: A framework for developing measurement systems and its industrial evaluation. Information and Software Technology 51(4), 721–737 (2009)
19. Torrents, V.R.: Development and optimization of a web-enabled OLAP-based software dashboard, Master thesis, Universidad de Alcala (2013)
20. Watson, H.J., Wixom, B.H.: The current state of business intelligence. Computer 40(9), 96–99 (2007)