Towards Automated Data Integration in Software Analytics

Silverio Martínez-Fernández
Fraunhofer IESE
silverio.martinez@iese.fraunhofer.de

Petar Jovanovic
Universitat Politècnica de Catalunya, BarcelonaTech
petar@essi.upc.edu

Xavier Franch
Universitat Politècnica de Catalunya, BarcelonaTech
franch@essi.upc.edu

Andreas Jedlitschka
Fraunhofer IESE
andreas.jedlitschka@iese.fraunhofer.de

ABSTRACT

Software organizations want to be able to base their decisions on the latest set of available data and the real-time analytics derived from them. In order to support "real-time enterprise" for software organizations and provide information transparency for diverse stakeholders, we integrate heterogeneous data sources about software analytics, such as static code analysis, testing results, issue tracking systems, network monitoring systems, etc. To deal with the heterogeneity of the underlying data sources, we follow an ontology-based data integration approach in this paper and define an ontology that captures the semantics of relevant data for software analytics. Furthermore, we focus on the integration of such data sources by proposing two approaches: a static and a dynamic one. We first discuss the current static approach with a predefined set of analytic views representing software quality factors and further envision how this process could be automated in order to dynamically build custom user analysis using a semi-automatic platform for managing the lifecycle of analytics infrastructures.

CCS CONCEPTS

• **Software and its engineering → Maintaining software**;

KEYWORDS

Data integration, real-time enterprise, ontology, software analytics

ACM Reference Format:
Silverio Martínez-Fernández, Petar Jovanovic, Xavier Franch, and Andreas Jedlitschka. 2018. Towards Automated Data Integration in Software Analytics. In *International Workshop on Real-Time Business Intelligence and Analytics (BIRTE '18), August 27, 2018, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3242153.3242159

1 INTRODUCTION

Nowadays, the huge amount of data available in companies has increased their interest in applying the concept of "real-time enterprise"[1] by using up-to-date information and acting on events as they happen. In this paper, we envision automated support for the real-time enterprise concept for software organizations by means of applying recent approaches to facilitate data integration tasks.

Currently, software organizations want to be able to base their decisions on the latest set of available data and the real-time analytics derived from them. The software development process produces various types of data such as source code, bug reports, check-in histories, and test cases [23]. The data sets not only include data from the development (e.g., GitHub with over 14 million projects), but also millions of data points produced per second about the usage of software (e.g., Facebook or eBay ecosystems). All this data can be exploited with "software analytics", which is about using data-driven approaches to obtain insightful and actionable information at the right time to help software practitioners with their data-related tasks [9]. This improves information transparency for diverse stakeholders. Bearing this goal in mind, we integrate these different data sources as a necessary first step in making this data actionable for decision-making. The integration becomes necessary because the inherent relationships in the data influencing the overall software quality are not obvious at first sight.

Despite its key role, the integration of different software analytics data still presents challenges due to the heterogeneity of the data sources. Not only do data come from sources carrying different types of information, but the same information is also stored in heterogeneous formats and tools. Big Data analytics involves the ingestion of real-time operational data into large repositories (e.g., data warehouses or data lakes), followed by the execution of analytics queries to derive insights from the data, with the final goal of performing business actions or raising alerts [6]. In a recent systematic review, data integration and final data aggregation were reported as part of the remaining challenges in Big Data analytics [19]. At the same time, another review in software analytics reported that most of the current approaches are still analyzing only one artifact [2], thus not focusing on integrating data from different sources and getting a holistic view. Thus, further research is needed to facilitate the integration of data sources for software analytics driven by the real information needs of end users.

To overcome the heterogeneity of software analytics data coming from different sources, we follow an ontology-based data integration approach in this paper; in particular, we intend to contribute: (a) the definition of an ontology capturing the semantics of relevant data for software analytics; (b) a current static approach for the integration of heterogeneous data sources given a set of predefined analytic views representing software quality factors; and (c) an envisioned approach for the dynamic integration of heterogeneous

[1] https://www.gartner.com/technology/research/data-literacy/

This is an Author's Accepted Manuscript of a paper to be published by ACM in the 12th International Workshop on Real-Time Business Intelligence and Analytics (BIRTE '18) at VLDB'18. The final authenticated version will be available online https://doi.org/10.1145/3242153.3242159

software analytics data, guided by the specific analytical needs of end users (i.e., information requirements).

The paper is structured as follows: Section 2 presents the related work. Section 3 presents an ontology for software analytics. Section 4 presents the implementation of a static approach to implement the integration. Section 5 discusses how the integration could be done dynamically. Finally, Section 6 concludes the paper.

2 BACKGROUND AND RELATED WORK

2.1 Software analytics and software quality

Contrary to the availability of data and its transparency in open source software, tool support for data integration for private companies in commercial systems is just emerging. For instance, we can find some large-scale software companies with their own proprietary development environments, such as Codemine (a proprietary software analytics platform) [7], Codebook (a framework for connecting engineers and their work artifacts) [4] by Microsoft, and Tricorder (a program analysis platform aimed at building a data-driven ecosystem) [18] by Google. Still, these platforms are proprietary and not widely used by others. In addition, companies like Kiuwan, Kovair, and Tasktop have recently started offering software and services for software analytics[2]. Furthermore, very recent research tools are CodeFeedr [21] and Q-Rapids [16]. Despite these efforts, there are still challenges for companies developing commercial systems to understand how to integrate heterogeneous data sources for software analytics.

Regarding software quality and quality models, a multitude of software-related quality models exist that are used in practice, as well as classification schemes [13]. One example is the ISO/IEC 25010 standard [1], which determines which quality aspects to take into account when evaluating the properties of a software product. A more recent example is the Quamoco quality model [22], which integrates abstract quality aspects and concrete quality measurements. Nowadays, having operationalized quality models offering actionable analytics from different data (system, process, and usage) is still a challenge.

2.2 Automating data integration tasks

Data integration is a well-studied area aiming at facilitating transparent access to a variety of heterogeneous data sources for the end user [15]. To automate data integration tasks, the use of Semantic Web technologies has been proposed [3]. In particular, we are interested in the use of an ontology for capturing the semantics of heterogeneous data sources due its machine-readable format and the inference capabilities it provides [5]. The automatic creation of data integration flows has been studied from two main perspectives: 1) starting by analyzing the available data sources (i.e., supply-driven; e.g., [11]), and 2) starting from the analytical needs of end users and further mapping them to the available data sources (i.e., demand-driven; e.g., [10]). Others also deploy a hybrid approach combining the previous two ideas [8, 12].

In this paper, we also envision a hybrid approach for integrating data sources related to software analytics, taking into account the real analytical needs of end users, which may vary over time.

3 INTEGRATING DATA SOURCES FOR SOFTWARE ANALYTICS

In this section, we present our software analytics use case that we will use throughout this paper.

3.1 An ontology for software analytics

We introduce an ontology that captures the semantics of relevant data for software analytics (see Fig. 1). In the ontology, each class represents an entity of the software analytics domain. For instance, the class *Issue* represents the issues from issue tracking systems. The ontology is abstract in order to enable generalization and applicability in different software projects. Therefore, the technologies used could differ among projects, whereas the concepts are present in most software development projects. For instance, for issue tracking systems, different companies may use different tools (e.g., Redmine, Jira, or GitLab), but all of them use issue tracking systems as a software development practice. Note that several approaches also propose automated generation of a domain ontology from the desired data sources to support data integration tasks (e.g., [20]).

The classes of the ontology in Fig. 1 represent data coming from the system either at development time or at runtime. For the sake of simplicity, we omit further ontology details (e.g., datatype properties) in Fig. 1, but explain the main process of how the ontology is built. During software development, we find data about the project and the development, which can be mapped, respectively, to the topics of improving software development process productivity and software system quality presented by Zhang et al. [23]. At runtime, we find data about the system behavior and its usage, which can be mapped to the topic of improving the software users' experience.

First of all, the project owner assigns *Persons* to a certain *Project*, by assessing their *Tech. Experience* with *Technologies.* When the project starts, the *Issues* are defined in an issue tracking system.

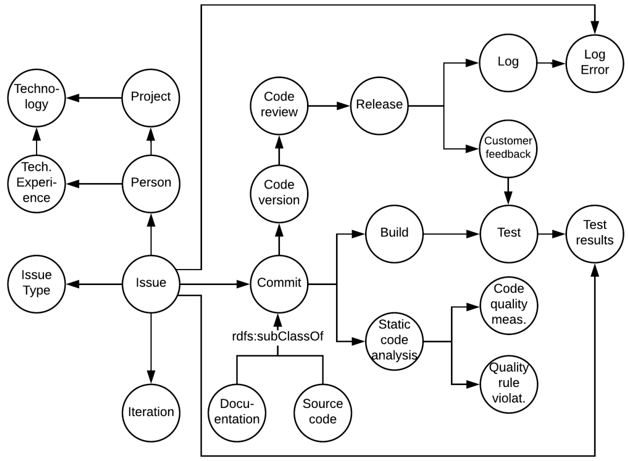


Figure 1: An ontology capturing the semantics of relevant data for software analytics

[2] Kiuwan (www.kiuwan.com), Kovair (www.kovair.com), Tasktop (www.tasktop.com)

During sprint planning, the product owner indicates the issues to be resolved in the current *Iteration* (a.k.a. sprint), and estimates the effort with story points. Therefore, it is known to which iteration(s) an issue is assigned. After sprint grooming and planning, the team works on issues, sending *Commits* (a.k.a. changes).

Commits include *Documentation* (such as how-to documents or architectural descriptions) and/or *Source Code*. After a commit is performed, automatic module and system *Builds* containing *Tests* are triggered. Therefore, the *Tests Results* can be associated with a *Commit*. Also, after a commit, *Static Code Analysis* is automatically triggered, producing *Code Quality Measures* (e.g., cyclomatic complexity, lines of code) and checking *Quality Rule Violations*. *Code Reviews* are done over a *Code Version* (a.k.a. branch) instead of intermediate results (i.e., single commits), requiring the results of automatic build and tests, as well as code quality measures and quality rules violations from static code analysis. This way, we know under which conditions a code review is accepted or rejected (e.g., thresholds for code quality measures or test results). If the commit is accepted, it is moved to the main line of a source *Code Version*. The approved source code is heavily tested in nightly builds (including stress testing and stability testing).

After successful nightly builds, the product owner may decide to create a new *Release*. When the new release is in use, runtime data becomes necessary: *Customer feedback* (normally relating to bugs and system functionalities) and *Logs* (from both end users and executed tests).

Besides the classes, the ontology contains meaningful associations among its classes (see arrows in Fig. 1). For instance, *Commits* can be integrated with *Issues* when the description of the *Commits* includes the "issue id" of the related issues.

3.2 Information requirements for end users

Analyzing information from single classes is not sufficient for reasoning about quality aspects of the software system, process, or usage. Previous research shows that relevant quality aspects (e.g., maintainability and reliability) contain metrics from heterogeneous data sources, i.e., from several classes of the ontology [17]. For this reason, we propose information requirements that consider several classes and help to reason over multiple metrics impacting a quality aspect. Below, we give two examples of information requirements.

Information requirement 1 (*IR1*). *Analyze the last release of the software product, per module, ordered by changes, quality rule violations, code quality measures (e.g., complexity, comments, and duplications).* The goal of IR1 is to improve the code quality of the modules in the next release. The different modules are studied with respect to changes, violations, and code metrics as measurements. Examples of action points to be taken by product owners are: allocating time in the next release to reduce the quality rule violations in a module, or deciding to refactor a highly changed module to make it more stable.

Information requirement 2 (*IR2*). *Examine the reliability of a release of the software product in terms of bugs found during testing and errors occurring at runtime, ordered by their resolution time.* The goal of IR2 is to improve the bug detection of tests. Examples of action points are: improving the test coverage of a unit test, creating further unit tests for a buggy module, testing the software in different contexts, and canceling a release with more bugs than the previous one.

The next two sections report two approaches for data integration using IR1 and IR2 as examples, respectively.

4 SOFTWARE ANALYTICS: CURRENT STATIC APPROACH

This section explains an implementation for real-time data integration done in the Q-Rapids tool [16]. The data flow can be summarized in three steps.

First, during data ingestion, data is gathered during development and at runtime from different data sources. This raw data is ingested into the data stores modeled on the basis of the ontology depicted in Fig. 1 and implemented as Elasticsearch indexes. As an example, for IR1, data is gathered from static analysis tools and version control systems and mapped to the following classes: *Code quality measure, Quality rule violation, Commit, Release*.

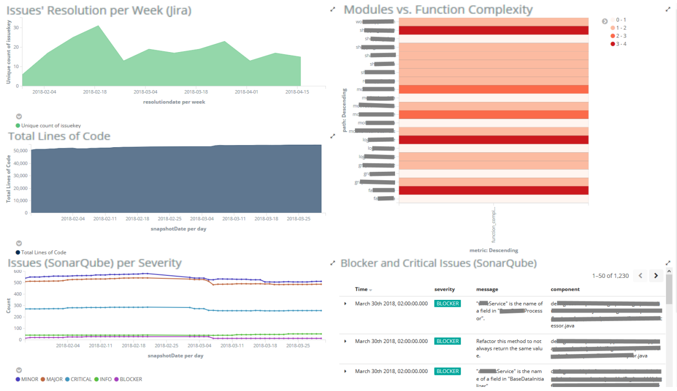
The high velocity at which data is coming into the system requires the use of Big Data technologies[3] for ingestion and analysis [16]. For this reason, the data from each source is ingested with a customized Apache Kafka connector, where the data source is the producer and the connector is the consumer. Then the data is pushed to an index in Elasticsearch, which represents the class of the ontology. For instance, for *Quality rule violation*, there may be multiple connectors for the heterogeneous tools (e.g., SonarQube, CodeSonar, Coverity). The class *Quality rule violation* can be integrated with others such as *Person* (author of the line of code violating a rule), *Code version* (component and line of code fields), and *Commit* (date field), among others.

Second, during data integration and aggregation, two activities are performed to enable the subsequent generation of alerts. In the first activity, quality metrics are computed from the ingested data and further interpreted with a value from 0 to 1. For instance, the assessed metric ' '*Fulfillment of critical/blocker quality rules*" gives a percentage of the files in the source code without critical or blocker quality rule violations (see [17] for details). When the necessary assessed metrics from several data sources are computed, they are stored in another data store as different Elastic indexes. Then the second activity starts. The assessed metrics are aggregated into product factors according to their weight. The weight indicates the relative importance of the assessed metric for the product factor. For instance, for *IR1*, the following assessed metrics are needed: *Non-complex files, Fulfillment of critical/blocker quality rules*, and *Highly changed files*. Following the example, these assessed metrics are aggregated into the *Code quality* and *Blocking code* product factors, as defined in the Q-Rapids quality model [17].

For the implementation, the predefined assessed metrics are translated into Elastic queries, including the formula as well as the execution frequency. Then the query pipeline is executed to compute the assessed metrics and product factors.

Third, product factor alerts can be raised for several reasons, such as a 'bad' value or prediction of a (normalized) product factor. Therefore, alerts act as traffic lights for product factors, only redirecting stakeholders to predefined dashboards when needed. These

[3] Apache Kafka for ingesting (https://kafka.apache.org/), Elasticsearch and Kibana of the Elastic stack for storing and visualizing (https://www.elastic.co/products)



Time	severity	message	component
March 30th 2018, 02:00:00.000	BLOCKER	"[redacted]Service" is the name of a field in "[redacted]Processor".	[redacted]essor.java
March 30th 2018, 02:00:00.000	BLOCKER	Refactor this method to not always return the same value.	[redacted]r.java
March 30th 2018, 02:00:00.000	BLOCKER	"[redacted]Service" is the name of a field in "BaseDataInitializer"	[redacted]

Figure 2: Example of raw data visualization for IR1

dashboards contain real-time data to support end users in solving the quality aspect monitored by the product factor.

These dashboards are implemented as Kibana objects (following Kibana terminology, they are dashboards consisting of visualizations and searches). As an example of IR1, an alert for *blocking code* can be raised because this product factor has been decreasing in the last sprints. Then the user is redirected to the dashboard shown in Fig. 2, with the following data about the selected release:

- "Modules vs. Function Complexity": It shows a heat map with the average function complexity of the modules. Users can drill down and see a list of files that should be checked.
- "Issue Resolution per Week" and "Total Lines of Code": By showing the velocity of the development team based on the resolution of issues and the evolution of the size of the code, the users can see if they are correlated with *blocking code* problems.
- "Issues per Severity" and "Blocker Issues": This shows the quality rule violations according to severity. In addition, a list of blocker and critical violations is shown that we suggest should be taken care of. Within this list, the user can filter the violations (e.g., rules of the type "code smell"). Also, by opening them, the user sees the module, line of code, and an explanation of the code smell problem. Therefore, after an alert has been raised, the user can see the details of each violation in order to take actions in real time.

5 TOWARDS AUTOMATED SOFTWARE ANALYTICS

In the previous section, we saw how the integration of data from multiple sources can be beneficial for extracting actionable analytics for the software process, system, and usage. However, as can be seen, such integration requires considerable manual effort on the part of the designer to integrate, transform, and prepare the data to be plugged into the desired analytics or visualization tool for further exploitation. Furthermore, given that information requirements may often be ambiguous and/or incomplete, the above process may undergo several rounds of reconciliation and redesign until the real information needs of an end user are finally met.

Considering that the analytical needs of different stakeholders, such as team leaders, project managers, or developers, are different and can, moreover, change dynamically over time, the proposed manual process may become an overburdening bottleneck.

Therefore, we envision a system that would apply and extend existing approaches in order to automate the process of building data integration flows for the field of software analytics. In particular, one such system, called Quarry [12], provides an automated, end-to-end solution for assisting the users of various technical skills in managing the design and deployment of analytical infrastructures, i.e., target data stores and data-intensive flows like extract-transform-load (ETL) processes. Quarry starts from high-level information requirements (e.g., IR1 and IR2 in Section 3.2) and semi-automatically generates target data stores and a data flow to integrate data from different data sources and prepare them for further exploitation.

For instance, a user interested in examining the reliability of a release in terms of issues of the type bugs found during testing and errors occurring at runtime detected in logs (see IR2 in Section 3.2) would pose such a requirement by selecting the ontological concepts in the graphical tool and adding additional query information.

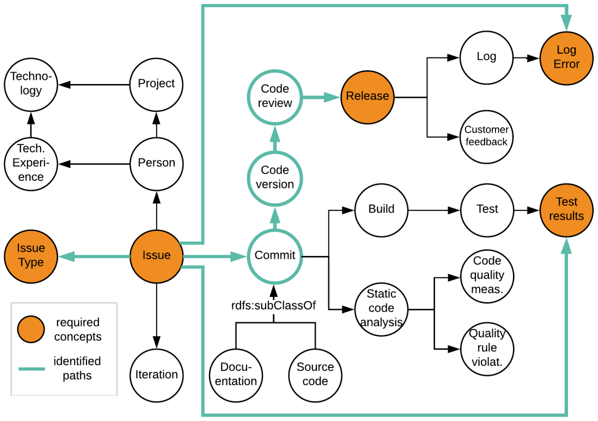


Figure 3: Identified ontological concepts (i.e., classes and properties) for IR2

Looking at the ontology in Fig. 1, the user can easily express such a requirement in his/her own vocabulary using Quarry's graphical tool, and hence Quarry will identify the ontological concepts requested by the user (i.e., *Release*, *Issue*, *Issue Type*, *Test results*, *Log Errors*; see Fig. 3). Moreover, the user can express restrictions regarding the given concepts (e.g., issue being of the type "bug") and aggregations (e.g., count the issues). Starting from the identified concepts, Quarry explores the ontology and finds the paths through which such concepts can be related. Notice that in order to satisfy the summarizability properties [14], which are needed to correctly answer the posed requirement, these paths must respect multidimensional integrity constraints (i.e., have "to-one" cardinalities). Thus, the selected paths are shown in Fig. 3. Going from here, Quarry extracts the subset of ontology concepts needed to answer the given requirement (see Fig. 4(a)), and generates the schema for the target data store (Fig. 4(b)).

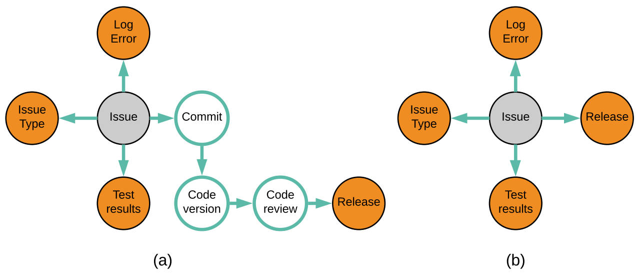


Figure 4: (a) Extracted ontology subset, and (b) generated schema of the target data store

In the background, Quarry also generates a complete data flow at the same time, which (1) extracts data from different data sources to which the identified ontology concepts map; (2) integrates these data following the identified paths in the ontology (by means of performing joins among the mapped data); and, finally, (3) applies restrictions (e.g., *Issue Type* = "*bug*") and aggregations (e.g., *count*(issues)) that the user may have expressed through his/her requirement.

To make all this work, Quarry also provides a deployment module, which can be extended to translate the generated constructs into the desired formats ready for deployment. On the one hand, the target data store schema can be translated either into a set of standard database tables implementing relational OLAP or into a set of Elasticsearch indexes as seen in the previous section. On the other hand, a data flow can be represented either as an ETL process implemented as a set of SQL views or in a proprietary format of an ETL tool (e.g., Pentaho Data Integration PDI), or as a query for immediately retrieving the data (e.g., SQL or Elasticsearch).

Finally, by deploying the generated analytical infrastructure (i.e., target schema and corresponding data flow), the system is ready to integrate and transform the input data coming from different data sources and to store it following the schema model in order to satisfy the user's information needs and prepare the data for further exploitation (e.g., real-time data visualization; see Figure 2).

6 CONCLUSIONS AND FUTURE WORK

The automatic integration of data from different sources is still a challenge for the software analytics domain. In this position paper, we defined an ontology capturing the semantics of software analytics data sources. Furthermore, we showed the current static approach to data integration in the Q-Rapids tool [16]. Finally, we envisioned a dynamic approach for the generation of dashboards with actionable analytics defined by the end users.

In the dynamic approach, the end users could, based on their own analytic needs, easily build data integration flows in order to prepare software analytics data to be plugged to external analytical tools. This will enable end users to explore and understand holistic software quality aspects by transparently considering different sources of information.

An immediate future direction is to conduct a detailed case study on applying automated approaches like Quarry [12] to the software analytics use case, with the aim of validating the benefits envisioned in this position paper.

ACKNOWLEDGMENTS

We thank Axel Wickenkamp for the implementation of the Q-Rapids tool and Sonnhild Namingha for proofreading the paper. This work was supported by the Q-Rapids project (H2020, No. 732253) and the ERCIM Fellowship Programme.

REFERENCES

[1] ISO/IEC 25010. 2011. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models. https://www.iso.org/standard/35733.html
[2] Tamer M. Abdellatif, Luiz F. Capretz, and Danny Ho. 2015. Software Analytics to Software Practice: A Systematic Literature Review. *BIGDSE@ICSE*, 30–36.
[3] Alberto Abelló, Oscar Romero, Torben Bach Pedersen, Rafael Berlanga, Victoria Nebot, María J. Aramburu, and Alkis Simitsis. 2015. Using Semantic Web Technologies for Exploratory OLAP: A Survey. *IEEE Trans. Knowl. Data Eng.* 27, 2 (2015), 571–588.
[4] Andrew Begel, Khoo Yit Phang, and Thomas Zimmermann. 2010. Codebook. *ICSE*, 125–134.
[5] Diego Calvanese, Martin Giese, Dag Hovland, and Martin Rezk. 2015. Ontology-Based Integration of Cross-Linked Datasets. In *ISWC*. 199–216.
[6] Badrish Chandramouli. 2015. Building Engines and Platforms for the Big Data Age. In *BIRTE@VLDB*. 23–37.
[7] Jacek Czerwonka, Nachiappan Nagappan, Wolfram Schulte, and Brendan Murphy. 2013. CODEMINE: Building a software development data analytics platform at Microsoft. *IEEE Software* 30, 4 (2013), 64–71.
[8] Umeshwar Dayal, Malú Castellanos, Alkis Simitsis, and Kevin Wilkinson. 2009. Data integration flows for business intelligence. In *EDBT*. 1–11.
[9] Harald Gall, Tim Menzies, Laurie Williams, and Thomas Zimmermann. 2014. Software Development Analytics. *Dagstuhl Reports* 4, 6 (2014), 64–83.
[10] Paolo Giorgini, Stefano Rizzi, and Maddalena Garzetti. 2005. Goal-oriented requirement analysis for data warehouse design. In *DOLAP*. 47–56.
[11] Mikael R. Jensen, Thomas Holmgren, and Torben Bach Pedersen. 2004. Discovering Multidimensional Structure in Relational Data. In *DaWaK*. 138–148.
[12] Petar Jovanovic, Oscar Romero, Alkis Simitsis, Alberto Abelló, Héctor Candón, and Sergi Nadal. 2015. Quarry: Digging Up the Gems of Your Data Treasury. In *EDBT*. 549–552.
[13] Michael Kläs, Jens Heidrich, Jürgen Münch, and Adam Trendowicz. 2009. CQML scheme: A classification scheme for comprehensive quality model landscapes. *EUROMICRO* (2009), 243–250.
[14] Hans-Joachim Lenz and Arie Shoshani. 1997. Summarizability in OLAP and Statistical Data Bases. In *International Conference on Scientific and Statistical Database Management*. 132–143.
[15] Maurizio Lenzerini. 2002. Data Integration: A Theoretical Perspective. In *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. 233–246.
[16] Lidia López, Silverio Martínez-Fernández, Cristina Gómez, Michał Choraś, R. Kozik, L. Guzmán, A. M. Vollmer, X. Franch, and A. Jedlitschka. 2018. Q-Rapids Tool Prototype: Supporting Decision-Makers in Managing Quality in Rapid Software Development. In *CAiSE Forum*. 200–208.
[17] Silverio Martínez-Fernández, Andreas Jedlitschka, Liliana Guzmán, and Anna-Maria Vollmer. 2018. A Quality Model for Actionable Analytics in Rapid Software Development. In *Euromicro SEAA 2018*.
[18] Caitlin Sadowski, Jeffrey Van Gogh, Ciera Jaspan, Emma Söderberg, and Collin Winter. 2015. Tricorder: Building a program analysis ecosystem. *ICSE*, 598–608.
[19] Uthayasankar Sivarajah, Muhammad Mustafa Kamal, Zahir Irani, and Vishanth Weerakkody. 2017. Critical analysis of Big Data challenges and analytical methods. *Journal of Business Research* 70 (2017), 263–286.
[20] Rizkallah Touma, Oscar Romero, and Petar Jovanovic. 2015. Supporting Data Integration Tasks with Semi-Automatic Ontology Construction. In *ACM Eighteenth International Workshop on Data Warehousing and OLAP*. 89–98.
[21] Enrique Larios Vargas, Joseph Hejderup, Maria Kechagia, Magiel Bruntink, and Georgios Gousios. 2018. Enabling Real-Time Feedback in Software Engineering. In *ICSE (NIER)*. 21–24.
[22] Stefan Wagner, Andreas Goeb, Lars Heinemann, Michael Kläs, Constanza Lampasona, Klaus Lochmann, Alois Mayr, Reinhold Plösch, Andreas Seidl, Jonathan Streit, and Adam Trendowicz. 2015. Operationalised product quality models and assessment: The Quamoco approach. *Information and Software Technology* 62 (2015), 101–123.
[23] Dongmei Zhang, Shi Han, Yingnong Dang, Jian Guang Lou, Haidong Zhang, and Tao Xie. 2013. Software analytics in practice. *IEEE Software* 30, 5 (2013), 30–37.