Visualizing Metric Trends for Software Portfolio Quality Management

Patric Genfer[*], Johann Grabner[†], Christina Zoffi[†], Mario Bernhart[†] and Thomas Grechenig[†]
[*] University of Vienna, Research Group Software Architecture, Vienna, Austria
Email: patric.genfer@univie.ac.at
[†] Research Group for Industrial Software, TU Wien, Vienna, Austria
Email: {johann.grabner, christina.zoffi, mario.bernhart, thomas.grechenig}@inso.tuwien.ac.at

***Abstract*—Software portfolios of today's companies are comprised of a variety of heterogeneous, modular, and often polyglot software solutions. Ensuring high-quality standards across these entire portfolios raises additional challenges for quality engineers and requires new strategies and visualization approaches to support software quality management decisions. In this paper, we study the information needs of software quality engineers that drive these decisions on portfolio-level and propose *Portfoliotrix* - an expert visualization prototype for a portfolio-wide analysis and comparison of software quality metric trends. For this, we first introduce our four-phase fully automated data-mining process for gathering and aggregating quality metric data from the underlying software repositories. We then present our application frontend that implements different visualization and filtering concepts to satisfy eight specific information needs derived from our prior qualitative semi-structured expert interviews. A final scenario-based expert evaluation confirms the practical relevance of our prototype. Experts took less time to complete quality-related management tasks with our tool than they had estimated when relying only upon existing toolsets. Moreover, our visualization system reached the System Usability Scale (SUS) score of "good" with 76.7 points.**

***Index Terms*—Data visualization, software metrics, software portfolios, software quality, trend analysis**

I. Introduction

While in past years managing software portfolios had mainly been considered a matter of larger enterprises [1], the topic is also gaining traction for mid-size or smaller companies recently. With the increasing acceptance of microservice architectures [2] and the growing usage of software product lines as application platforms [3], today, even smaller businesses can benefit from the advantages of heterogeneous software landscapes [1]. In addition, even larger companies have been steadily expanding their portfolios, either through organic growth of their software products [4] or, notably in the last decade, by taking over competitors together with their software solutions[1]. Regarding these growing portfolio sizes, some aspects need to be considered on a much larger scale.

One of these issues, which is also the main subject of our work, is the process of analyzing and managing the quality within software portfolios, for which we use the term *Software Portfolio Quality Management* (SPQM). While quality management is a vast area on its own, here, we understand SPQM as *any decision-making activity that ensures and supports a common quality standard and process across multiple related software artifacts*. We further assume that the relationship shared by the software is arbitrary and ranges from divergent development branches of a single application through product lines up to all applications of an organization.

SPQM at this portfolio scale can create some particular challenges software quality experts have to face. According to Elonen and Artto [5], one major challenge for stakeholders is to dissect the relevant information for decision-making from the pool of available data, which becomes even more difficult for larger portfolios where the amount of collected data can quickly reach a significant size. If the gathered data is not filtered or aggregated appropriately, this can lead to a potential information overflow, resulting in inadequate or even wrong decisions [5]. Nevertheless, to identify the information relevant for quality managers, it would first be necessary to define their specific information needs regarding software portfolio quality precisely. However, previous research in this area has mainly focused only on single repositories (see, e.g. [6, 7]) and not on SPQM as a whole. This limitation is also the case for the majority of commercial quality platforms, most notably SonarQube[2], which plays a dominant role in this section.

While most solutions provide a feature-rich experience for tracking and visualizing various quality metrics for single repositories, they show only limited support for handling large software portfolios. But especially this lack of possibilities to trace quality standards on portfolio level is a huge deficit that needs to be addressed, especially if one considers the growing numbers of projects within portfolios. Having a convenient way of tracking portfolio-specific quality metrics and analyzing their trends over a longer period would allow quality experts to adjust their resources better [1] and adapt their overall project strategies to achieve the best possible results [4]. To bring such a holistic portfolio monitoring approach to life, a new kind of expert visualization tool would be necessary, one that addresses the concrete requirements of portfolio-level quality analysis. However, to realize such a visualization system, some hurdles have to be overcome:

- In contrast to single repositories, portfolios can be very polyglot and host applications of various technologies and

[1]https://www.visualcapitalist.com/interactive-major-tech-acquisitions/

[2]https://www.sonarqube.org/

use cases [8], making it challenging to define an overall quality standard suitable for all projects.

- Projects within the portfolio can have different development and release cycles, making it difficult to compare their quality aspects over a more extended period.
- Combining various quality metrics from several software projects can quickly lead to information overload. This information must be reduced to a reasonable level without losing too much detail.

All these aspects must be considered when providing a holistic solution for SPQM. The purpose of this paper is, therefore, to develop *Portfoliotrix* (from *Portfolio* M*etrics*), a prototypical expert visualization system for monitoring and analyzing quality metrics on the software portfolio level. In this context, we also aim at answering the following research questions:

RQ1: *What are the information needs in software portfolio quality management?* To discover these information needs, we conducted qualitative semi-structured expert interviews after a systematic literature review. Thus, we combined the current state of the art with our vision and the knowledge from domain experts.

RQ2: *How does a visualization satisfy these information needs?* Utilizing the findings of the previous research question, we performed a requirement analysis and specification to derive and rank requested visualization features. Then, an iterative agile software engineering process followed that concluded with the implementation of *Portfoliotrix*, our expert visualization system prototype.

RQ3: *How satisfied are experts with the proposed visualization?* We evaluated our prototype with a scenario-based expert evaluation accompanied by a quantitative questionnaire about task complexity and relevance. At last, we let the experts rate the visualization usability and debriefed them with an open discussion.

After discussing the related work, this paper has dedicated one section for each research question above in the same order. The remaining Sections VI and VII discuss the results of our study and conclude with an outlook on further research efforts in this area.

II. Related Work

While there are several studies about the information needs of software developers and managers (see [6, 7, 9, 10, 11]), most of them focus on different aspects than portfolio quality management. Buse and Zimmermann [6], for instance, studied the information needs of over 100 software and lead developers at Microsoft. While their research concentrates on single software repositories, some of their findings can also be adapted on the portfolio level. Jedlitschka et al. [7] categorized the information needs of software managers, but their focus was more on identifying the demands that drive the decision process for selecting appropriate development technologies. In this context, the research of Elonen and Artto [5] and Kuipers and Visser [4] are some of the few studies that address the concrete challenges software managers encounter when managing software portfolios. While the first mentioned authors identified a set of problem areas portfolio managers have to face, the latter ones describe an iterative tool-based process to gather and analyze portfolio quality metrics on different levels.

As another essential part of portfolio quality management, data visualization has also been the subject of several studies. A general overview of different data visualization techniques is provided by Keim [12] and Grant [13]. Both introduce different visualization techniques and provide various use cases for their application. Kienle and Müller [14] present a comprehensive study regarding the functional and non-functional requirements a visualization tool must meet. Although their investigation aimed mainly at research-related tools for monitoring software quality and maintenance, their findings can easily be applied to our SPQM approach. A more portfolio-specific study was conducted by Staron et al. [15]: The authors derived some structural and functional recommendations a quality project dashboard should ideally meet, based on their examination of three real-world companies' development process. Many of these considerations influenced the development process of our visual components.

Most of the aforementioned studies deal with the topic in a more observational sense and do not include any prototypical implementations. In contrast, Sakamoto et al. [16] implemented *MetricsViewer*, a concrete frontend client application, as part of their approach to collect quality metrics from source code repositories. While their prototype focuses on monitoring the evolution of single repositories, the *Empirical Project Monitor* introduced by Ohira et al. [17] allows, albeit to a limited extent, the combination of metrics from different projects into a single view.

III. Information Needs in Software Portfolio Quality Management

Besides their decision-making in software quality, engineers may also fulfill other roles in their organization [18]. Therefore, it is not easy to differentiate the target group of quality managers from other quality-related positions like software developers or architects [6]. We thus have not limited ourselves to a specific job position but instead considered all roles with responsibility for quality-related tasks as relevant for addressing **RQ1** in this section.

According to our definition of SPQM, we initially collected relevant information needs and problems from the scientific literature. Our problem selection criteria were that their cause is an underlying unsatisfied information need and that visualizations can solve them. For these problems, we uncovered their related information needs.

The study in [5] reports known and uncovers additional problems that arise while managing project portfolios. From these issues, we derived the following information needs (IN):

IN1: *Personalized Detail Levels:* Project managers, software architects, and developers need information at the detail level of their tasks, but also the possibility to seamlessly navigate between these different detail levels. If tools neglect this requirement, the consequence is either information overload or lacking data quality.

IN2: *Data Actuality, Consistency, and Centralization:* Since software comprises many source code and configuration changes that immediately affect quality, the analysis data must stay updated. Infrequent monitoring leads to wrong conclusions about the current status. Furthermore, the data acquisition method must be equal across the portfolio, or the projects lose their comparability. Finally, decision-makers require centralized data for overviews and comparisons. Fragmented and distributed data hinder their work and make it more error-prone.

The symptoms of inadequate portfolio management compiled in [1] enabled us to infer two more new information needs:

IN3: *Early Problem Notifications:* The decision-makers need early notifications about low-quality conditions. If undetected, these situations have an unconfined negative impact on the project. Moreover, worsening quality leads to escalations that require unplanned reallocations of project resources.

IN4: *Shared Quality Guidelines:* Objective decisions on the portfolio level need to be backed by transparent and shared quality guidelines. Otherwise, differing standards across projects become incomprehensible on the portfolio level.

Based on our conclusions from the scientific literature, we designed a semi-structured expert interview with qualitative and quantitative questions to answer our first research question. The goal of our interview was to discover new information needs and validate the existing ones. After a pilot session to improve the completeness and clarity of our questions, we conducted three interviews. According to our SPQM definition, we selected three experts - two engineers and one software architect - from a software research and development organization with over 300 employees. In this study, we asked our participants to explain their answers with reasons and relied on our logical assessment of their soundness.

While finding more evidence for the derived information needs, we also discovered new ones in our interviews:

IN5: *Historical Quality Trends:* For our interviewees, the history of quality metric changes over a project duration was as relevant as the current state. Being aware of ongoing downward trends enables the early prevention of severe conditions before they arise.

IN6: *Health Indicators with Thresholds and Exceptions:* As the data density of quality metrics in a portfolio quickly causes information overload, our experts expressed the need for an aggregated quality indicator per project. Furthermore, configurable metric thresholds would assist them in applying a shared quality standard across the portfolio. However, they deemed making occasional exceptions with configurable thresholds per project necessary for rare circumstances.

IN7: *Project Comparison:* Although this need is evident, we learned that our experts are not satisfied with their current means of overseeing a portfolio and comparing different metrics across projects. What is missing for them is a configurable side-by-side view across projects that promotes exploration and comparison while presenting their most important quality metrics at a glance. The combination of different metrics was also rated as an essential feature by our experts, as single metrics are often not expressive enough, a conclusion also supported by literature (see [19]).

IN8: *Code Proximity:* Our experts called for actionable tool support while monitoring portfolios. Next to the process and documentation, source code is the main contributor to software quality. In this context, actionable means that portfolio monitoring needs to uncover the causes for quality changes in the source code itself. Knowing the cause makes deriving concrete counter-measures with a high positive impact on quality straightforward.

Due to our questioning, we also explored the background information and rationales behind our interviewee's answers. When asked about the usefulness of different quality metrics representations, participants favored statistical distributions first, followed secondly by minimum and maximum aggregations, and the commonly used mean and median at last. With this quality data exposed, it would be easier for them to fight technical debt, measure the effectiveness of their quality measures, and communicate the condition of projects to other stakeholders. A portfolio quality visualization would also enable them to assess and compare different frameworks and libraries. At last, noteworthy is that they see the quality analysis necessary per single commit because the combined impact of multiple commits makes it hard to identify the causes.

IV. Software Portfolio Quality Visualization

To satisfy the information needs mentioned above and answer **RQ2**, we examined various visualization approaches and combined them into our prototype. This section gives an overview of our implementation and how the concepts used serve the specific information needs and requirements identified in Section III.

A. Data Mining Process

Before serving the portfolio quality metrics to individual stakeholders, a four-phase fully automated data-mining process, as described in Figure 1, extracted and calculated the relevant quality metric data from the underlying source code repositories. The collected code artifacts and the computed metric values were then interlinked and stored in a document-based graph database[3] for later processing through the visu-

[3]https://www.arangodb.com/

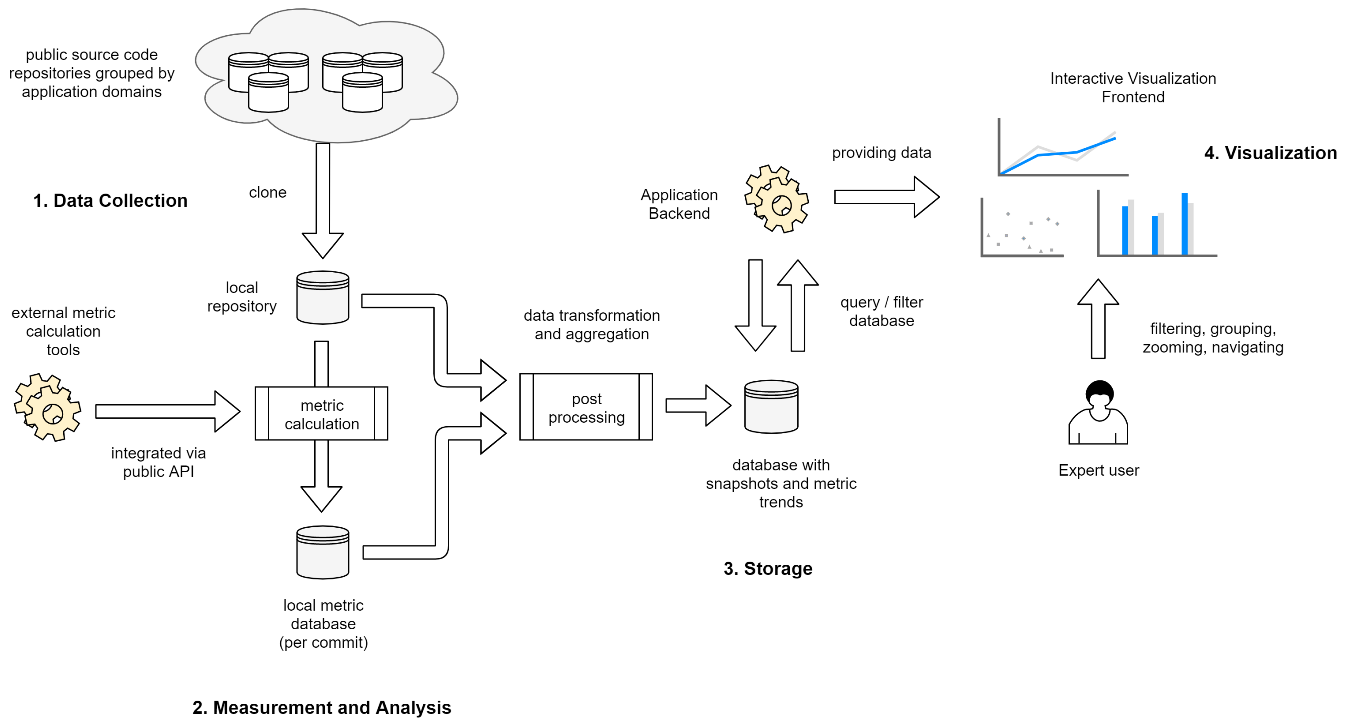

public source code repositories grouped by application domains
1. Data Collection
clone
local repository
external metric calculation tools
metric calculation
integrated via public API
local metric database (per commit)
2. Measurement and Analysis
data transformation and aggregation
post processing
database with snapshots and metric trends
3. Storage
Application Backend
query / filter database
providing data
Interactive Visualization Frontend
4. Visualization
filtering, grouping, zooming, navigating
Expert user

Figure 1. Metric Visualization Process

alization client. Aggregate functions like *Average*, *Median*, or *Sum*, but also metric distributions per commit were calculated upfront to improve rendering speed at runtime and thus provide a better user experience [14].

Our data-mining process supports GitHub[4] repositories as sources but can easily be configured to support other hosters as well. The same applies to the external metric calculation tool used during the second execution step. Currently, Understand[5], a third-party quality metric suite, is preconfigured but integrating other metric calculators that provide other, more portfolio-specific metrics is also possible. For our prototype, we relied on commonly used static OOP and complexity metrics provided by Understand, like *Lack of Cohesion of Method* (LCOM), *Coupling between Object Classes* (CBO) or *Cyclomatic Complexity* (CC) [20]. Incorporating other, more runtime-related metrics would have also been possible, but as compiling and running repositories can be time-consuming and error-prone [21], this was outside the scope of this work. While some initial manual configuration steps, like setting up the repository access and integrating the external metric calculator, are necessary, the actual data mining process runs fully automated. Extracting the relevant repository commits and calculating the selected metric values is achieved without any human interaction, making this approach especially well suited for DevOps or Continuous Integration scenarios, allowing a consistent portfolio analysis (see also ***IN2*** in this context).

[4]https://github.com/
[5]https://www.scitools.com/

B. Portfolio Dashboard

Portfoliotrix's initial application screen, the *Portfolio Dashboard* in Figure 2, provides a comprehensive overview of all portfolio projects (2) and allows stakeholders to get a quick overview of the overall portfolio (***IN1***, ***IN2***). As 2D line charts are considered a common and familiar way of presenting historical time series [13], they were used throughout the whole dashboard for visualizing the quality metric trends. By default, every project metric is rendered into a single diagram (4) to prevent potential cognitive overload (*overplotting* [14]). However, users can also combine several metrics into a single trend chart to compare trends more efficiently (3) as demanded by information needs like ***IN5*** and ***IN7***. In addition, users can choose from three different aggregate functions (7) to visualize different metric trends. While all metrics support *Median* and *Average* value representations for all artifacts of a commit, some specific ones, like *Lines of Code* (LOC), also provide a cumulative display. *Portfoliotrix* also supports the option to switch the representation from absolute values to relative changes starting at the initial value to facilitate a more straightforward comparison of metrics with different value ranges [13].

C. Health Indicator

As many experts mentioned during the interviews, getting a fast overview of the overall quality state, together with a notification system in case of quality violations, is crucial functionality for them (***IN3***, ***IN4***, ***IN6***). Therefore, our frontend provides the possibility to define and fully customize

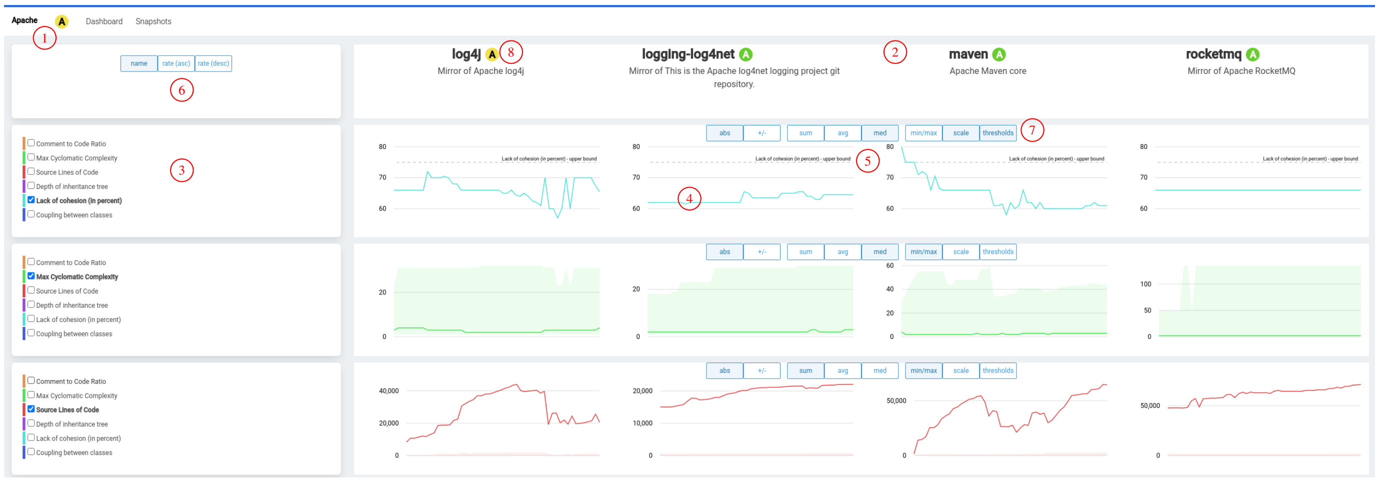
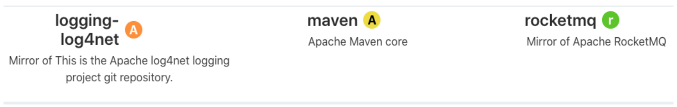


Figure 2. *Portfoliotrix* Dashboard View with the following elements: (1) selected portfolio and overall health state, (2) list of projects within portfolio, (3) list of selected metrics, (4) trend visualization of selected metrics per project, (5) metric threshold, (6) sort order of projects, (7) controls for adjusting trend graph visualization, (8) individual project health state

threshold-based health indicators. Quality experts can define individual lower and upper thresholds for every metric, either individually per project or on a global portfolio-wide scope. The threshold is visualized through a horizontal indicator line in the respective diagram chart (5). An iconic display [12] further encodes the overall quality status: A circular symbol next to the project's name (8) expresses any rule violation using a four-color code similar to a traffic light [15]. A red-colored indicator symbolizes that the metric trend has recently violated the threshold, while a green circle background means that all parameters are within the optimal range. If the indicator changes to orange, then the trend is not yet violating the quality rule, but it is constantly worsening. On the other side, a yellow color shows that the trend is near the critical value but improving.

The last two color indicators can also be considered as an early warning system, allowing experts to react before the actual quality violation happens [15]. The overall portfolio state (1) arises from the highest project-specific rule violation (unless the rule is explicitly excluded from the overall state). In that way, a user can immediately recognize whether any quality rule is violated without investigating each project separately.


logging-log4net A
Mirror of This is the Apache log4net logging project git repository.

maven A
Apache Maven core

rocketmq r
Mirror of Apache RocketMQ

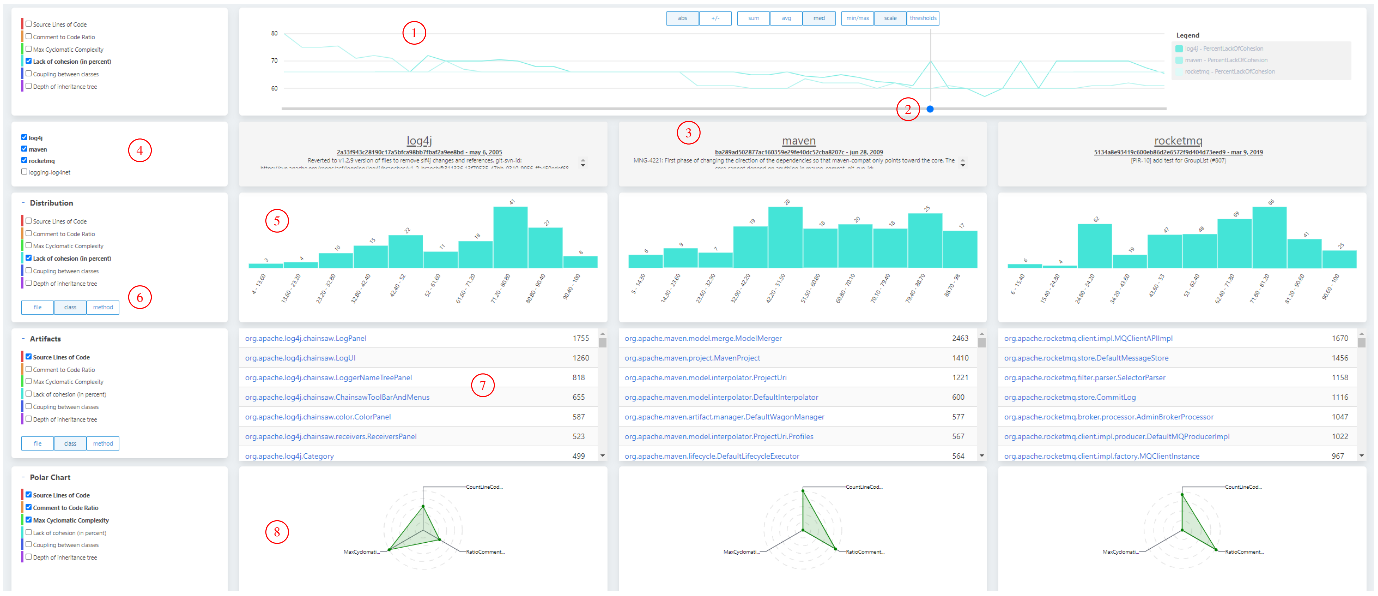
Figure 3. Custom Health Indicators. While the first two projects use the same quality rules, recognizable through the letter *A* in their health indicator circle, the last one uses a project-specific indicator *r*.

D. Snapshot View

In addition to the dashboard, users also have access to a view that provides detailed information per snapshot/commit. This *Snapshot View* in Figure 4 combines different visualization concepts, each focusing on a specific quality aspect, and by providing a much finer level of detail than the dashboard view, it focuses mainly on the needs of software architects and engineers (see ***IN1***). The view's root element is a trend graph similar to the ones used in the dashboard view, but with the additional option to combine several metrics as well as multiple projects within a single diagram (1). Besides, this chart also provides a vertical slider control (2) that allows users to navigate back and forth through the portfolio's entire history (***IN5***). The layout of the remaining view organizes the selected projects into separate columns, each column containing a header cell with additional commit metadata (3) and various visualizations. Users can freely choose which projects they want to incorporate into this view (4), allowing for a better direct comparison, as requested by some experts (***IN7***).

The second cell presents the distribution of the recorded metric values in a histogram (5). The frequencies are summarized in buckets, with the number of elements per group displayed above each category. Since histograms contain even quite dense multidimensional data when used only for a single metric, combining several metrics into a single chart would significantly increase the cognitive load [13]; therefore, users can only select a single metric at once for visualization. To limit the information density further, filtering code artifacts by their granularity level, either *file*, *class*, or *method* (6), is also possible.

The following row, the *Artifact List* (7), provides the most granular access to the underlying source code artifacts within the whole application and gives users direct access to source code (***IN8***). Studies have also shown that accessing all relevant information like metrics and source code through a single application can improve developers' efficiency during work [22], as they do not have to switch between different applications to connect metric values with their corresponding artifacts. Therefore, this visual component shows all tracked



org.apache.log4j.chainsaw.LogPanel	1755
org.apache.log4j.chainsaw.LogUI	1260
org.apache.log4j.chainsaw.LoggerNameTreePanel	818
org.apache.log4j.chainsaw.ChainsawToolBarAndMenus	655
org.apache.log4j.chainsaw.color.ColorPanel	587
org.apache.log4j.chainsaw.receivers.ReceiversPanel	523
org.apache.log4j.Category	499

org.apache.maven.model.merge.ModelMerger	2463
org.apache.maven.project.MavenProject	1410
org.apache.maven.model.interpolator.ProjectUri	1221
org.apache.maven.model.interpolator.DefaultInterpolator	600
org.apache.maven.artifact.manager.DefaultWagonManager	577
org.apache.maven.model.interpolator.ProjectUri.Profiles	567
org.apache.maven.lifecycle.DefaultLifecycleExecutor	564

org.apache.rocketmq.client.impl.MQClientAPIImpl	1670
org.apache.rocketmq.store.DefaultMessageStore	1456
org.apache.rocketmq.filter.parser.SelectorParser	1158
org.apache.rocketmq.store.CommitLog	1116
org.apache.rocketmq.broker.processor.AdminBrokerProcessor	1047
org.apache.rocketmq.client.impl.producer.DefaultMQProducerImpl	1022
org.apache.rocketmq.client.impl.factory.MQClientInstance	967

Figure 4. *Portfoliotrix* Snapshot Centric View with following elements: (1) combined trend graph for selected projects, (2) slider to navigate to individual snapshots, (3) meta data of selected snapshots, (4) list of projects selected for visualization, (5) distribution of selected metric for current snapshot, (6) artifact filter, (7) list of metric values per artifact, (8) polar chart with normalized metric values

artifacts together with their recorded metric values. A click on a specific list entry lets the user jump directly to the hosted artifact's source code in the GitHub repository. Analogous to the histogram component, this view also provides filter options to narrow down the type of artifact.

The final visualization component uses polar charts (8) to let the user combine several metric values into a single diagram. In this chart, every metric gets a separate axis, and the value ranges are normalized from zero to one over the whole project lifetime. In that way, users can compare the relation of selected metrics to each other at a specific point in time to answer, for instance, questions like "*While the lines of code have doubled between two commits, how did the documentation ratio develop during this time*? ". Another advantage of this kind of visualization is that specific metric combinations build reoccurring shapes, which makes it easier to detect specific architectural anti-patterns [23]. While this view can provide some meaningful cross-project insights, its relatively high information density requires a certain degree of expert knowledge about the structure and characteristics of individual projects and therefore is more suited for advanced scenarios.

V. Evaluation

For answering **RQ3** of how satisfied experts are with our visualization prototype, we conducted a scenario-based expert evaluation. We built our evaluation and structure of this section upon the approach presented in [24]. After going through the evaluation methodology, this section reports the participant demographics, the task scenarios, and the results.

A. Methodology

Initially, we created a plan to define our goals, scenarios, questionnaires, and the procedure for our scenario-based expert evaluation. For answering our third research question, the goals were to collect demographic participant data, provide practically relevant scenarios, compare our visualization to existing tools, and learn about its perceived usability. Due to the need for social distancing because of the COVID-19 pandemic, we designed the evaluation procedure for conducting it remotely in an online meeting with Zoom[6].

As a portfolio to visualize, we chose a small subset of projects[7] from the Apache Software Foundation. As downloading and analyzing the relevant commits of all repositories and generating the highly connected metric-artifact database afterward takes a considerable amount of time, we defined the condition for selecting a project as the conjunction of three criteria to retain resource and cost-efficiency. The projects should have less than 200 KLOC, use Java or C# as a language, and be popular on GitHub. Applying these criteria resulted in RocketMQ[8], Maven[9], Log4j[10], and log4net[11] as our evaluation data set. Table I summarizes the project selection criteria.

Especially the comparison between log4j and log4net, where the latter is a port of the first one to a different language, provides interesting insides regarding the use of different technologies for solving the same problem.

[6]https://zoom.us/
[7]https://projects.apache.org/projects.html
[8]https://github.com/apache/rocketmq
[9]https://github.com/apache/maven
[10]https://github.com/apache/log4j
[11]https://github.com/apache/logging-log4net

Table I
SELECTED ASF PROJECTS USED FOR EVALUATION

Repository	Description	KLOC	Stars	Analyzed commits
log4j	Java based logging framework	127	704	1,5 %
log4net	C# port of log4j	129,4	459	4,7 %
Maven	project management and build tool	191	2,1 K	0,5 %
RocketMQ	distributed messaging platform	173	10,9 K	3,75 %

Like in our expert interviews, we ran one pilot session to identify critical problems with the visualization prototype, task descriptions, and the evaluation process itself. After the incorporation of the findings, we proceeded to our six evaluation sessions. Three out of the six participants had already taken part in our past interviews for **RQ1**. From the same organization as before, we selected three software engineers with SPQM activities as new participants.

Our evaluation sessions started with an introduction to explain the procedure and the visualization context to every participant. With the participant's consent, we began recording the video and audio of our online conference. The participant got access to the testing environment through screen-sharing with remote control, where we had prepared three browser tabs containing the questionnaire, a cheat sheet, and the visualization prototype. After a participant had answered the demographic questions, we provided her or him additional time to consult the digital cheat sheet that summarized the main features of our visualization. The sheet was also available to every individual throughout the whole session. When the participant was ready to go on, we continued with the task scenarios.

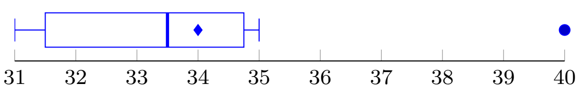
Each participant had a unique order of scenarios because of our counterbalancing against learning effects with a Latin square [25]. Since we had seven scenarios for six participants, we packed the closely related scenarios C and D into one Latin square element. Thus, these two scenarios were always in consecutive order.

Before using our visualization to solve each scenario, the participants had to rate the difficulty and estimate the time investment of solving it with their familiar toolset. We encouraged all participants to be vocal about their thoughts while giving their answers and using our visualization. As test moderators, our predefined criteria for intervening in a scenario were when we discovered a participant's misunderstanding of the task or when our visualization showed unexpected behavior. The participants had to solve the tasks self-sufficiently, and interventions from our side according to our rules were rare. To compare our participants' effort estimations with our visualization, we measured the task completion times. Once our participants came up with a solution, we asked them to rate the practical relevance of the task to their work.

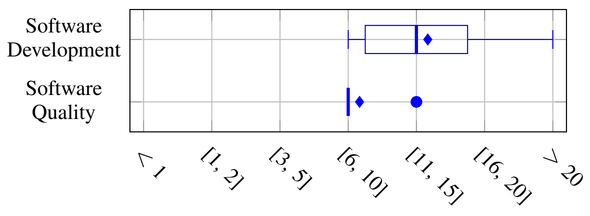
After working through all scenarios, the participants had to rate the user interface's suitability and information density and fill out the System Usability Scale (SUS) [26] form. Finally, we concluded each test session with a debriefing in the form of an open interview where we asked the participants for their impressions and thoughts about the visualization.

B. Participant Demographics

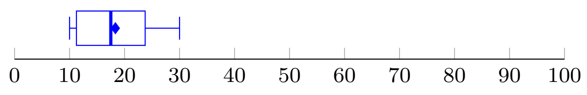
We selected our participants across different project teams of the same organization, and all of them performed SPQM activities as part of their work. The participants' average age was 34 years. Out of the six people, one identified as female and the remaining five as male. Their organization considered all of them senior software engineers. On average, they had 11 to 15 years of experience in software development and 6 to 10 in software quality activities. They estimated their daily investment in software quality as an average of 18 percent of their working time. Figure 5 summarizes the demographics of our participants.



(a) *Participants' age in years*



(b) *Participants' experience in years per topic*



(c) *Participants' estimated percentage of SPQM activities in their work*

Figure 5. Demographics of the participants

C. Task Scenarios

The following list documents the tasks that our participants had to complete in our evaluation sessions. Additionally, the list provides titles to disclose the intended idea behind every task. However, we did not show these titles to our participants because this knowledge provides unwanted hints at the scenarios' solution processes.

(A) *Comparing metric trends*: Regarding Maven, do the changes of the average Lines of Code metric correlate stronger with the changes of the average Comment to Code Ratio or with the average Cyclomatic Complexity metric?

(B) *Determining the health state of the portfolio and its projects on lower detail level*: Regarding their latest trend, which projects have a median lack of cohesion (LCOM) that is higher than 65?

(C) *Determining the health state of the portfolio and its projects on higher detail level*: Regarding its latest trend, which project has an artifact (file or class) that has a maximum cyclomatic complexity of more than 50?

(D) *Define individual health criteria per project*: Keep the portfolio state to green even if RocketMQ does violate the previously defined cyclomatic complexity rule.

(E) *Identify single classes within the whole portfolio*: Regarding Maven and RocketMQ, which is the highest class coupling value a class has and which class is it?

(F) *Comparing projects on different detail levels*: What might be a possible reason that the total lines of code of log4j and log4net became more similar?

(G) *Monitoring metrics over time*: Regarding the Maven project, how does the amount of classes with one base class and classes with two base classes change between May 15, 2009 and Jun 20, 2020?

D. Results

Our evaluation results consist of the participants' task and usability ratings, their qualitative feedback, and the recorded task completion times.

1) Task Scenario Rating and Completion Times: Before solving a task, our participants rated its difficulty and estimated how long they would need to solve it with their familiar toolset. After completing each task, they judged how relevant it is to their SPQM activities. For all three ratings, we employed a quinquepartite scale. The difficulty and relevance rating had a scale of attributes ranging from the lowest to the highest value. The effort rating utilized a scale of duration intervals from less than five minutes to over an hour.

Figure 6 shows our participants' ratings and our measurements in detail. The participants rated tasks F and G as the most difficult of all. In comparison, the other ratings tilted to a lower difficulty. Effort estimation ratings showed a similarity to the assessed difficulty. Regarding the relevance, we received answers with averages close to the value of 4 for each scenario, which means the majority of our participants rated the tasks as either relevant or very relevant for their daily work. For comparing participants' effort estimates with their actual time investment, we measured the scenario completion times. The average time was always under 4 minutes. In most cases, the experts solved the scenarios by a margin below their estimations. Furthermore, they also completed the two most demanding tasks significantly under their expectations.

2) Visualization Rating and System Usability Scale (SUS): As another part of our evaluation, we investigated the usefulness and usability of our visualization prototype. After the participants had worked through the scenarios, we asked them to rate the suitability and information density of the user interface. Figure 7 shows that the experts largely agreed that the interface was well suited for the tasks while the amount of information was still manageable. Up next, we presented the frequently used System Usability Scale [26] form to our participants. In Figure 8, we show the detailed results. Our prototype scored an average of 76.7 points on this scale, which translates to the attribute of "good" usability. Three noteworthy points on which all experts highly agreed were the system's consistency, the well-integrated feature set, and a personal low-entry barrier for using the system.

3) Qualitative Feedback: While our participants worked through the scenarios and in our concluding debriefing at the end, they gave us qualitative feedback on the prototype and voiced their associative thoughts on SPQM.

The experts received the visualizations of the statistical distributions of quality metrics well. They acknowledged this feature as an advantage over their familiar tools. There was even an interest to see how these distributions change over time to be more aware of quality trends. The visualization should actively aid in making distribution changes over time more transparent. Switching between consecutive snapshot views is not enough to draw comparisons conveniently.

The participating experts also wished for more customizability in the visualization. Their feedback repeatedly indicated that they want dynamic views to add and remove different visualization elements freely to suit their use cases. We heard a broad range of different ideas ranging from supporting more quality metrics to dynamically constructing custom views with the available visualization components. These ideas also suggested that the components should be more customizable and come with more interaction methods. Zooming, panning, and also resizing were requested features. The wish for more customizability also showed up in the feature request for saving and loading configurable views and reports, especially for the dashboard area.

As another additional interaction method, the experts wished for more filtering options. The experts also deemed the times relevant when the project activity happened, so there should be filter options for specific time intervals. Although our scope was on the portfolio level, an expert wanted to have project-specific filtering options that take the source code structure into account. This type of filtering would add value on the portfolio level too, when multiple projects have similar components like front- and backends.

Another category of feedback was that the system should also act as an assistant. Before counter-measures can address quality problems in a portfolio, somebody has to discover the issues first. Therefore, the system should actively notify about problems and thereby increase quality awareness.

Lastly, minor usability issues surfaced due to the apparent prototype status of our visualization system. The experts discovered contrast and alignment issues and called for a proper description of all axes and labeling improvements. Despite being resolved quickly, these issues showed to have a considerable negative impact on usability.

E. Threats to Validity

This section gives an overview of some common threats to validity (see also [27]) we encountered during our research and how we addressed them.

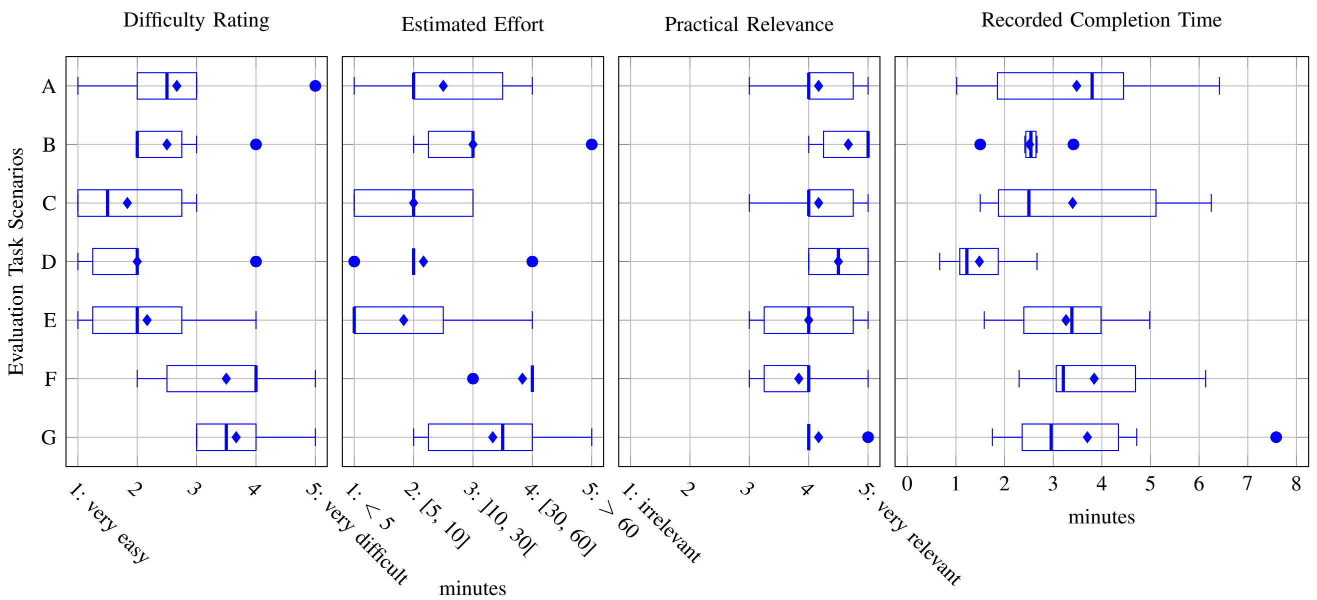


Figure 6. Participants' rating of the task difficulty, estimated effort, and practical relevance compared with the recorded completion time

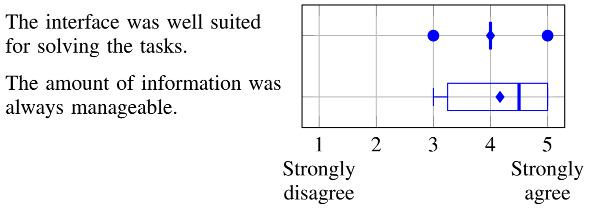
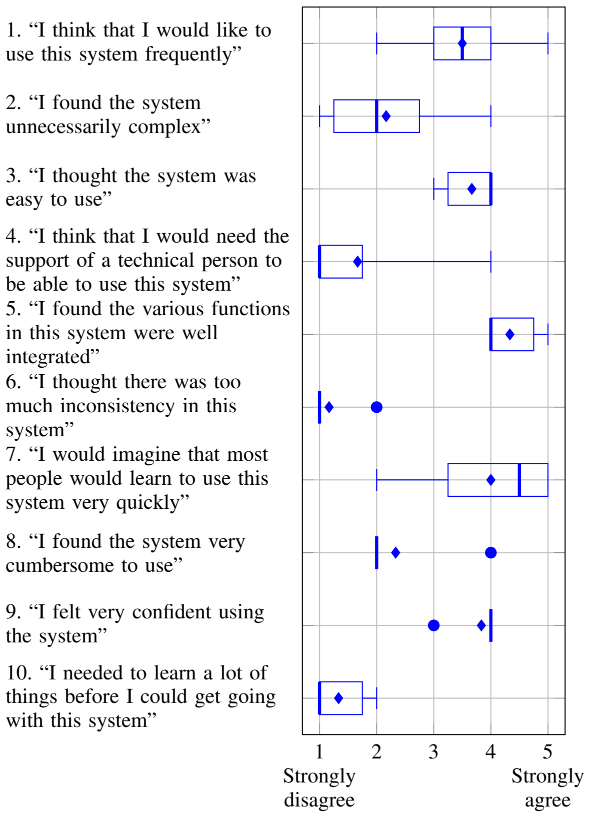
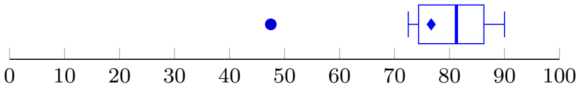


Figure 7. Participants' rating of the user interface and information density

- *Internal Validity* can be affected by unknown factors which can influence the evaluation results without being recognized by the researchers. Some of these threats we identified were possible learning effects when participants executed our scenarios. We tried to minimize this effect by providing each expert with a different order of the test scenarios. Furthermore, we should also note that all our participants were employed at the same company, resulting in a potentially biased test audience. To compensate for this bias, we still have to validate our insights with larger samples from different organizations.
- *Construct Validity* expresses how well our research approach and the corresponding results represent the real-world situation. We used a combination of time and complexity estimations and qualitative interview questions from our participants to benchmark our prototype with existing solutions. While each of these factors in isolation would not have been sufficient enough to draw any sound conclusions, using all factors together to evaluate the prototype might have reduced this threat. Nevertheless, additional measurement criteria would strengthen our results.



(a) *Responses to the items of the System Usability Scale*



(b) *Total scores on the System Usability Scale*

Figure 8. Results of the System Usability Scale [26] evaluation

- *External Validity* describes how well the gathered results could be generalized to a larger problem space. While we gathered a subjective impression of our prototype by a group of selected quality experts through our interviews, our results cannot be considered representative due to our small sample size. Also, our study revealed that the target group of software quality managers is much more inhomogeneous than our test audience group, and interviews on a much larger scale would be necessary for more representative results.

VI. Discussion

The interviews conducted for **RQ1** revealed that apart from appropriate tools, SPQM as an activity is currently not yet sufficiently addressed in practice, although the experts concede a clear added value through organization-wide cross-portfolio quality standards. If carried out at all, SPQM is not installed as an explicit role but rather carried out by experts from different fields of activity as part of their work. This is seen as an interesting finding and calls for a broader investigation of information needs of the different individual roles involved. Furthermore, we identified a strong prevalence of SonarQube as a platform for quality-related activities among all three experts interviewed, which may have impacted the preferred metrics and functions.

As for **RQ2**, our visualization system has holistically shown how to incorporate the discovered information needs. However, a recurring theme in the expert's feedback was that they want the visualization to be more customizable and dynamic. Although our solution shows rudiments for flexibility by allowing the user to add more rows in the dashboard, most components and their layouts are still static. A logical remedy for this rigidity is to provide the current user interface elements as layout-responsive components that act as building blocks of a visualization construction kit. A preset would provide the current layout, and the users could build and rearrange their dashboards for different use cases while every component adapts its level of detail respective to its size.

Considering the measured results, the qualitative feedback and the SUS score, we deduce for **RQ3** that the experts perceived our prototype very positively and were satisfied with the visualizations and functions provided. When questioned about how well the individual functions were implemented in the prototype, five out of six participants rated at least four out of possible five points. In addition, also the overall information density of the visualizations provided was noticed highly manageable (4.2 on average) by the experts. The positive perception in conjunction with the high level of agreement on the practical relevance of the quality-related activities covered in the scenarios (on average between 3.8 and 4.7 for all scenarios) can be interpreted as validation and fulfillment of the prior derived information needs. The detailed values in Figure 6 show that experts were able to complete the majority of tasks significantly faster than they had initially estimated when relying upon existing toolsets. It was also interesting to see that experts rated some specific scenarios as very complex and estimated a much higher effort for solving them with existing toolsets, as they eventually required when using *Portfoliotrix*. For example, five out of six experts indicated 30 to 60 minutes for scenario F but completed it only within a few minutes (03:50 min on average) with our prototype. This observation leads to the assumption that many quality-related activities on portfolio-level are currently not sufficiently covered by existing tools and might have been misestimated by the experts in terms of difficulty and effort.

VII. Conclusion

In this paper, we examined the information needs of software quality engineers to ensure high-quality standards across entire software portfolios and support them with portfolio-wide quality management decisions. Using qualitative semi-structured expert interviews, we derived and validated four existing information needs drawn from literature and identified four additional new ones. Furthermore, we presented a fully automated, highly configurable, and extensible data mining process for calculating and aggregating quality metrics from source code repositories of a given software portfolio.

We then introduced *Portfoliotrix*, our prototype of an expert visualization system that addresses the prior determined information needs and allows monitoring and analysis of software quality metric trends on a portfolio level. We provided a holistic solution to track and compare project-specific quality metrics on a portfolio level, establish health indicators, and evaluate the effectiveness of applied quality measures.

A scenario-based expert evaluation with six software engineers - all with at least six years of experience in software quality assurance - confirmed the practical relevance of quality-related management activities facilitated by our prototype. Moreover, our evaluation showed that experts could complete specific tasks in significantly less time than they had estimated when solving the same tasks with existing toolsets. Finally, we reached a SUS score of "good" with 76.7 points which confirmed the usability of our prototype.

To conclude, the obtained information needs, together with the proposed visualization concepts, serve as a foundation for further research in this area. The interviews carried out in this paper can - due to the limited number of participants - only be regarded as an initial contribution. A broader analysis of information needs should be deeper explored in future studies. Furthermore, our current prototype visualizes solely quality metric data retrieved from source code repositories. However, today's software engineering projects also consist of database schemes, environments, build-and deployment configurations for DevOps processes, or even machine learning models, as additional essential artifacts of a holistic, cross-portfolio quality management process. Depending on the respective artifact, new metrics, quality analysis tools, and visualization concepts might also be necessary. Last, further research should also investigate expanding our prototype with novel functionalities. A portfolio-wide AI-based automated hotspot/point of interest detection mechanism would be another area that offers high potential for future work.

References

[1] J. Vahaniitty, K. Rautiainen, and C. Lassenius, “Small software organizations need explicit project portfolio management,” *IBM Journal of Research and Development*, vol. 54, no. 2, pp. 1:1–1:12, 2010.

[2] N. Alshuqayran, N. Ali, and R. Evans, “A systematic mapping study in microservice architecture,” in *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*. IEEE, 2016, pp. 44–51.

[3] J. Bosch and P. Bosch-Sijtsema, “From integration to composition: On the impact of software product lines, global development and ecosystems,” *Journal of Systems and Software*, vol. 83, no. 1, pp. 67–76, 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121209001617

[4] T. Kuipers and J. Visser, “A tool-based methodology for software portfolio monitoring,” in *Proceedings of the 1st International Workshop on Software Audits and Metrics - SAM, (ICEIS 2004)*, INSTICC. SciTePress, 2004, pp. 118–127.

[5] S. Elonen and K. A. Artto, “Problems in managing internal development projects in multi-project environments,” *International Journal of Project Management*, vol. 21, no. 6, pp. 395–402, 2003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0263786302000972

[6] R. P. L. Buse and T. Zimmermann, “Information needs for software development analytics,” in *Proceedings of the 34th International Conference on Software Engineering*, ser. ICSE ’12. IEEE Press, 2012, p. 987–996. [Online]. Available: http://dl.acm.org/citation.cfm?id=2337223.2337343

[7] A. Jedlitschka, N. Juristo, and D. Rombach, “Reporting experiments to satisfy professionals’ information needs,” *Empirical Software Engineering*, vol. 19, no. 6, pp. 1921–1955, 2014.

[8] I. Nadareishvili, R. Mitra, M. McLarty, and M. Amundsen, *Microservice architecture: aligning principles, practices, and culture*. O’Reilly Media, Inc., 2016.

[9] D. N. Card and C. L. Jones, “Status report: practical software measurement,” in *Third International Conference on Quality Software, 2003. Proceedings.*, 2003, pp. 315–320.

[10] A. Begel and T. Zimmermann, “Analyze this! 145 questions for data scientists in software engineering,” in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 12–23. [Online]. Available: https://doi.org/10.1145/2568225.2568233

[11] A. J. Ko, R. DeLine, and G. Venolia, “Information needs in collocated software development teams,” in *29th International Conference on Software Engineering (ICSE’07)*, 2007, pp. 344–353.

[12] D. A. Keim, “Information visualization and visual data mining,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 8, no. 1, pp. 1–8, 2002.

[13] R. Grant, *Data visualization: charts, maps, and interactive graphics*. Crc Press, 2018.

[14] H. M. Kienle and H. A. Müller, “Requirements of software visualization tools: A literature survey,” in *2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis*, 2007, pp. 2–9.

[15] M. Staron, W. Meding, J. Hansson, C. Höglund, K. Niesel, and V. Bergmann, “Chapter 8 - dashboards for continuous monitoring of quality for software product under development,” in *Relating System Quality and Software Architecture*, I. Mistrik, R. Bahsoon, P. Eeles, R. Roshandel, and M. Stal, Eds. Boston: Morgan Kaufmann, 2014, pp. 209–229. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780124170094000089

[16] Y. Sakamoto, S. Matsumoto, S. Saiki, and M. Nakamura, “Visualizing software metrics with service-oriented mining software repository for reviewing personal process,” in *2013 14th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, 2013, pp. 549–554.

[17] M. Ohira, R. Yokomori, M. Sakai, K.-i. Matsumoto, K. Inoue, and K. Torii, “Empirical project monitor: a tool for mining multiple project data,” *IET Conference Proceedings*, pp. 42–46(4), 2004. [Online]. Available: https://digital-library.theiet.org/content/conferences/10.1049/ic_20040474

[18] S. Komi-Sirvio, P. Parviainen, and J. Ronkainen, “Measurement automation: methodological background and practical solutions a multiple case study,” in *Proceedings Seventh International Software Metrics Symposium*, 2001, pp. 306–316.

[19] J. Rosenberg, “Some misconceptions about lines of code,” in *Proceedings Fourth International Software Metrics Symposium*, 1997, pp. 137–142.

[20] L. H. Rosenberg and L. E. Hyatt, “Software quality metrics for object-oriented environments,” *Crosstalk journal*, vol. 10, no. 4, pp. 1–6, 1997.

[21] H. Barkmann, R. Lincke, and W. Löwe, “Quantitative evaluation of software quality metrics in open-source projects,” in *2009 International Conference on Advanced Information Networking and Applications Workshops*, 2009, pp. 1067–1072.

[22] M. Lanza and S. Ducasse, “Polymetric views - a lightweight visual approach to reverse engineering,” *IEEE Transactions on Software Engineering*, vol. 29, no. 9, pp. 782–795, 2003.

[23] M. Pinzger, H. Gall, M. Fischer, and M. Lanza, “Visualizing multiple evolution metrics,” in *Proceedings of the 2005 ACM Symposium on Software Visualization*, ser. SoftVis ’05. New York, NY, USA: Association for Computing Machinery, 2005, p. 67–75. [Online]. Available: https://doi.org/10.1145/1056018.1056027

[24] J. Grabner, R. Decker, T. Artner, M. Bernhart, and T. Grechenig, "Combining and visualizing time-oriented data from the software engineering toolset," in *2018 IEEE Working Conference on Software Visualization (VISSOFT)*, 2018, pp. 76–86.

[25] J. V. Bradley, "Complete counterbalancing of immediate sequential effects in a latin square design," *Journal of the American Statistical Association*, vol. 53, no. 282, pp. 525–528, 1958. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/01621459.1958.10501456

[26] J. Brooke, "SUS - A quick and dirty usability scale," *Usability Evaluation in Industry*, vol. 189, no. 194, pp. 4–7, 1996.

[27] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. Springer Berlin Heidelberg, 2012.