Design and Validation of Precooked Developer Dashboards

Vladimir Ivanov
Innopolis University
Innopolis, Russia
v.ivanov@innopolis.ru

Vladislav Pischulin
Innopolis University
Innopolis, Russia
dalv6666@gmail.com

Alan Rogers
Innopolis University
Innopolis, Russia
a.rogers@innopolis.ru

Giancarlo Succi
Innopolis University
Innopolis, Russia
g.succi@innopolis.ru

Jooyong Yi
Innopolis University
Innopolis, Russia
j.yi@innopolis.ru

Vasili Zorin
Innopolis University
Innopolis, Russia
v.zorin@innopolis.ru

ABSTRACT

Despite increasing popularity of developer dashboards, the effectiveness of dashboards is still in question. In order to design a dashboard that is effective and useful for developers, it is important to know (a) what information developers need to see in a dashboard, and (b) how developers want to use a dashboard with that necessary information. To answer these questions, we conducted two series of face-to-face individual interviews with developers. In the first step we analyzed answers, build a Goal-Question-Metric model and designed a precooked developer dashboard. Then, during the second separate series of interviews, we validated the GQM and derived feedback on the designed dashboard. Given that the cost of dashboard customization prevents developers from utilizing dashboards, we believe that our findings can provide a solid starting point to build precooked developer dashboards that can be readily utilized by software companies.

CCS CONCEPTS

• **Software and its engineering → Software development process management**;

KEYWORDS

developer dashboards, interviews with developers, GQM method

ACM Reference Format:
Vladimir Ivanov, Vladislav Pischulin, Alan Rogers, Giancarlo Succi, Jooyong Yi, and Vasili Zorin. 2018. Design and Validation of Precooked Developer Dashboards. In *Proceedings of the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '18), November 4–9, 2018, Lake Buena Vista, FL, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3236024.3275530

1 INTRODUCTION

There is increasing interest and use of developer dashboards in the software engineering industry [4, 8, 25]. Developer dashboards are typically used to visualize the overall status of a project — the assumption here is that visualization helps managers/developers be aware of the overall status of the projects they are working on, and make proper collaborative/individual decisions while developing software, which will improve the overall productivity of a development team [8, 15, 25]. However, this promising assumption of a developer dashboard would hold, only when a dashboard displays information needed by the users, without distracting the users with unwanted information.

What kinds of information do developers want to see in a dashboard? While there would be no single answer to these questions, we seek to find general answers from software engineers in the field. Our findings can be used to construct a useful default dashboard. Although most modern dashboards are customizable, developers often use the default dashboard due to the cost entailed by customization — for example, developers often do not want to read through the full functionalities the dashboard provides, and consequently do not customize the dashboard, as reported in [25]. The same paper [25] reports that managers often do not customize dashboards either, and calls for a better default dashboard. Our work tackles this niche.

We conduct face-to-face individual interviews with representatives of 44 different companies. We designed our questionnaire/sessions, taking into account the Goal-Question-Metric (GQM) method [3].

Our key contributions in this paper are: *(1) to the best of our knowledge, we for the first time identify metrics developers want to see in dashboards, by employing an observational study [20], (2) provide a GQM model to understand why the identified metrics matter for the developers, and (3) identify how developers want to use dashboards in their development processes.* Given that the cost of dashboard customization prevents developers from utilizing dashboards [25] (our participants expressed similar responses), we believe that our findings can provide a solid starting point for software development organizations who seek to improve team productivity by utilizing developer dashboards. Last but not least, our validation process has revealed an additional aspect in requirements representation and visualization.

2 BACKGROUND AND RELATED WORK

The task of building a dashboard involves selecting the appropriate metrics for measuring the software product, as well as the minimum required set of functionality in the dashboard. One of the approaches to selecting the necessary metrics is the collection of

ESEC/FSE '18, November 4–9, 2018, Lake Buena Vista, FL, USA

ACM ISBN 978-1-4503-5573-5/18/11…$15.00
https://doi.org/10.1145/3236024.3275530

all possible measurements. The effectiveness of this method is questionable. First, data collection is a time-consuming process. Second, the question of interpreting the results remains open. Another option for metrics selection is based on the Goal-Question-Metric approach. This approach deliberately forces the screening of metrics by binding them to pre-established goals. This concept provides interpretation of the collected data and is the de facto standard in building a dashboard [5]. Thus, metric selection should be guided by the proper selection and prioritization of goals of dashboard's users. In this section we first review related work about what is the purpose of dashboards in software engineering and then analyze work that describes how to apply GQM in dashboards design.

2.1 Dashboards for software development

Dashboards increase collaboration and make the communication process more transparent [6, 25], since they spread information in the organization, helping to control project status, finding bottlenecks, increasing peripheral awareness, and comparing teams. Dashboards have also been found useful in distributed teams [13]. In essence, properly designed dashboards can be used as an extremely effective tool for visual control for the development of high-quality software products [14].

Substantially there are three basic types of dashboards: (1) strategic, (2) operational and (3) analytical [10]. In [14], the following two basic interaction modes of a dashboard is listed: pull and push. In the pull, the user wants to use a dashboard in order to get some specific information. In the push, a dashboard is used like a notification panel, which pushes highly important information to a user, triggering almost predefined followup actions.

The types of, and mode of interactions with dashboards are strongly connected with their specific purposes. Yigitbasioglu and Velcu [27] list the following four basic purposes for creating a dashboard: (1) performance monitoring, (2) measurement consistency, (3) communication, and (4) planning. Lastly, to collect the data for the dashboard there are mainly two approaches: manual [9] and automatic (non-invasive) [21].

2.2 Designing dashboards and GQM

At the end of this analysis, it should be clear that creating an effective and useful dashboard is not easy: stated simplistically it requires one to select the "right" data and the "right" visualization techniques.

The concept of "right" is indeed hard to define. A step toward the identification of such data and visualization techniques can be performed using the Goal-Question-Metrics approach [1]. This approach allows companies to define which data should be collected and how it can be interpreted. It provides a goal-based framework for software measurement.

In [22], the authors formed a GQM model for the dashboard, which focuses on the size, complexity, planning, cost and quality of the product being created, and validated it using a survey of 110 industry professionals using the Likert scale. Respondents rated the following metrics: Lines of Code (LOC), Function Points, McCabe's Cyclomatic Complexity. As a result, 74% of respondents consider these 8 metrics to be the basis for software product measurements, noticing for the fact that LOC is the least significant metric.

In [18] the author aims to develop a set of metrics for software development groups to find out what software development teams should measure to effectively monitor their progress. The work was conducted in the infrastructure of a real company with 9 agile-teams, through the collection of requirements, joint discussions of requirements for the dashboard functionality, and the subsequent use and evaluation of the created dashboard in real work.

The result of the work is 5 measurements in 3 different spheres, covering the most important needs of agile-team of software developers for effective monitoring of their progress. There are Number of open defects, Number of estimated and remaining story points, Number of planned, Successfully executed tests, Integration speed.

3 GENERAL STRUCTURE OF THE STUDY

The study reported in this article has two main stages (Fig. 1). We start with building a "typical" GQM needed for dashboard design by running a series of interviews with industry. Then, we validate the result using the same instrument with another set of companies. Figure 1 shows the key stages of the work. This diagram depicts 2 main stages: "GQM model generation" and "GQM model validation". The paper is structured in a similar way.

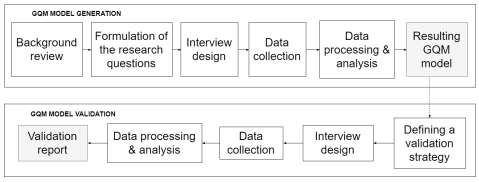


Figure 1: Structure of the study

3.1 Building a "typical" GQM

Through the first step we collected information from software engineers and developers to build a "typical" GQM, from which to derive the dashboard. There is one proviso: we did not pose explicit questions on company goals and strategies, to avoid being exposed, also unintentionally, to sensitive information; rather, we remained at a more generic level that allowed us to preserve full confidentiality while still collecting everything practically needed.

Designing an effective questionnaire for our interview is not easy, since the way questions are defined, organized, and laid down may significantly influence the overall result. Therefore, we followed the rules defined in [1, 2, 7, 26]. We selected a closed questionnaire using the techniques proposed by Furnham [11] and of Podsakoff et al. [19] to minimize the biases in the responses, including redundancy and replication. Moreover, we employed the GQM itself to design the questionnaire [1, 2]. In other words, we used the GQM approach to design a questionnaire aimed at defining a "typical" GQM [12]. Taking this approach, the **goal** of the investigation was to try to build a "typical" GQM used in software organizations, or a significant subset of it, and the "typical" visualization to use effectively in (several) companies not having the effort to build a fully fledged dashboard .

3.2 Validation of GQM

After building the GQM it need to be validated. The results were validated using a face-to-face interviews with another set of companies. It consists of 4 main parts:

- **part 1:** verifying the established relationships between the indicators and the goal, similar goals, dashboard features;
- **part 2:** checking availability of metrics in companies;
- **part 3:** reidentifying the goals of the development process;
- **part 4:** collecting demographic data.

Relationships between metrics and goals: in order to verify the relationship of metrics to the goal, a prototype of the dashboard and synthetic scenarios were used, one for each goal. Dashboard is implemented in a form of an HTML page with JavaScript insets for dynamic content changes (Fig. 2). The page contains 13 metrics selected for the study. Each metric is placed in a tile. Three metrics are represented as a graph, 10 metrics are represented as numerical values. On the page there is a slider that is responsible for the day of the iteration. Changing its position changes the state of metrics. This is necessary to keep track of the change history. Each tile has a color indication to indicate the acceptability of the state of the contained metric.

This prototype simulates the software development situation by a team of 5 people using the Scrum methodology, 10 iterations were completed, each iteration consists of 14 days. The task of this prototype is to find out whether the relationship between the metrics and the goal is obvious to a developer, and also to understand which metrics are the most representative for this relationship.

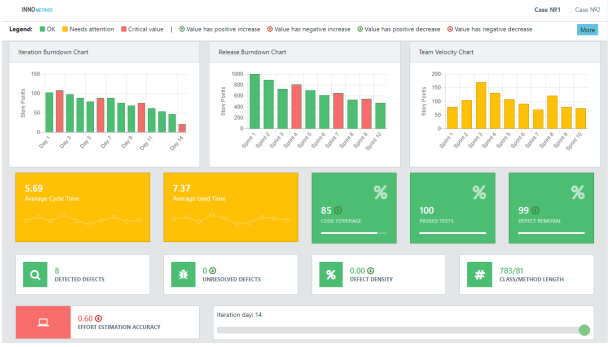


Figure 2: Dashboard interface for interview

Features of a dashboard: after a personal immersion in the interaction with the dashboard and the end of the answers to the questions of the first scenario, the respondent was given the opportunity to comment on his/her experience of interaction. The respondent was asked to comment on the interaction, point out the extra elements and what functionality he/she lacked. After this, the transition to the second scenario was carried out and after the completion of which the respondent was asked about the desire to supplement his thoughts on the dashboard.

Metrics availability: the main source of metrics is the tools used in companies. The Project Management Tool (PMT) is one of the main such supplier of quantitative indicators of the development process. Here we asked which company management tool is used.

Goals of the development process: to reidentify the existing goals (that are usually related to common issues of the development process), we asked this question directly. Questions about the current development cycle, the place of work, the communication team were asked earlier. We also asked questions about the validation of already identified goals.

Demographics: the fourth part is devoted to demographic issues; it helps to understand how the results can be generalized.

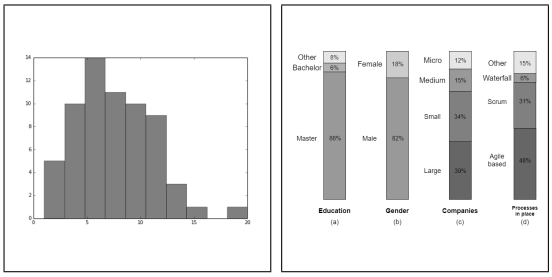
3.3 Administration of the instrument

To increase the external validity of our findings, we have followed the best practices suggested in the literature [16, 17, 24]. In particular, we drew interview participants from multiple different companies located in a high-tech city, and conducted a face-to-face interview with each participant. More specifically, the questionnaire was first sent to the participants, then a member of the research team interviewed them face-to-face, recorded the results in a written document, shared the written document back to the participants to ensure that the right information was captured, and corrected the results if required. Each interview lasted about one hour and the overall preparation for it and followup check required about two hours.

4 BUILDING "TYPICAL" GQM AND DEVELOPER'S DASHBOARD

4.1 Demographics

When building a "typical" GQM we have interviewed 44 companies. The working experience of respondents ranged from 1 to 20 years, with a median value of 7 (Figure 3(a)). Majority of them (89%) had at least a graduate degree. (Figure 3(b)): 86% hold a Master degree, 3% have a PhD, 6% are Bachelors, and 5% have no university education.



(a) Working experience of the participants (b) Profiles of the participants

Figure 3: Information about participants

Overall, the participants were homogeneously distributed across the participating companies and they appear a reasonable representation of the overall population of the developers and software engineers present in the area and around the world [23].

4.2 Resulting "typical" Goals

The majority (54%) answered that effort estimation is the biggest obstacle. While the effectiveness of effort estimation (that is, how effective the estimated effort is) can be measured with a metric

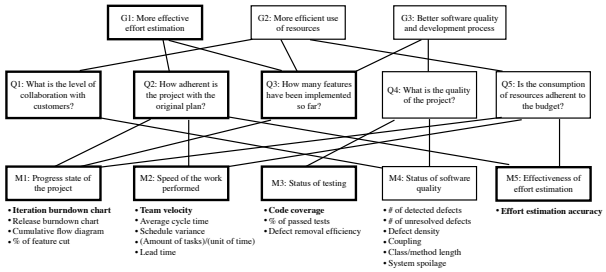


Figure 4: The resulting "typical" GQM model where strong labels and high-frequency metrics are depicted in boldface

(indeed, our GQM model contains effort estimation accuracy as one of its high-frequency metrics), effort estimation itself is currently conducted in a rather ad-hoc way, relying on the experience of the developers. Given this situation, we find strong need from the industry for more systematic data-based effort prediction—further research on this area is warranted. Overall, we summarize our first step results as follows:

Based on our results, we have extracted a "typical" GQM model (shown in Figure 4), consisting of three goals: more effective effort estimation **(G1)**, more efficient use of resources **(G2)**, better software quality and development process **(G3)**; five questions; five metric categories. We have also noticed that our informants show less interest in software quality than other kinds of metrics. Meanwhile, our informants showed the strongest interest in iteration burndown charts (81%).

4.3 Questions

We have identified the following "typical" questions – three strong questions and two moderate questions. We consider a question strong, if the question is identified based on a high-frequency answer.

Q1.What is the level of collaboration with customers? This question mainly reflects the importance of customer satisfaction (86%). This question also covers number of change requests (13%) and team velocity (36%), tasks with the biggest cycle time (12%).

Q2.How adherent is the project with the original plan? This question reflects the importance of delivering a product on time (71%), percentage of implemented features (73%), and amount of unimplemented tasks or user stories (75%). This question also covers schedule variance (40%) and number of overdue tasks (13%) and team velocity (36%).

Q3.How many features have been implemented so far? This question reflects the importance of implementation of features (58%), percentage of implemented features (73%), and amount of unimplemented tasks or user stories (75%), most critical defects (36%).

Q4.What is the quality of the project? This question covers quality-related responses such as meeting quality and safety standards (34%), quality metrics (27%), and most critical defects (36%). This question also covers reports of crashes and other problems (15%), percentage of passed tests (13%) and tasks with the biggest cycle time (12%).

Q5.Is the consumption of resources adherent to the budget? It covers budget-related responses such as staying within the budget supported (30%), over-budgeted tasks (18%), cost performance index (15%), cost variance (15%), and team velocity (36%).

4.4 Metrics

We have identified the five distinct kinds of metrics: (1) metrics to show the current progress state of the project, (2) metrics to show the speed of work performed, (3) metrics to show the status of testing, (4) metrics to show the status of software quality, and (5) metrics to show the effectiveness of effort estimation. We have also connected these metrics with the goals developers want to achieve, through a GQM model (Fig. 4). We have observed that, in general, developers are more concerned about monitoring whether the development process is on track than monitoring software quality. We believe that with this categorization, software engineers will be able to understand more easily what kinds of metrics other software engineers are typically interested in, than when they simply read through the list of metrics.

M1.Progress state of the project: This metric category mainly represents iteration burndown charts (81%) of Table 1. It also covers release burndown charts (28%) and cumulative flow diagram (13%) of Table 1, and percentage of features cut during the project (18%) of Table 3.

M2.Speed of the work performed: This metric category concerns the dynamic aspect of the software development process—that is, how fast progress is made over time. This category mainly represents team velocity (51%) of Table 1. Other metrics that can be included in this category are: average cycle time (10%) of Table 1, and schedule variance (39%), amount of tasks per unit of time (34%), and lead time (13%) of Table 3.

M3.Status of testing: This metric category mainly represents code coverage (67%) of Table 2. Other metrics that can be included in this category are: percentage of passed tests (31%) of Table 1, and defect removal efficiency (25%) of Table 3.

M4.Status of software quality: These metrics include number of detected defects (39%) and number of unresolved defects (36%) of Table 1; defect density (49%), coupling (16%), and method or class length (10%) of Table 2; and system spoilage (19%) of Table 3.

M5.Effectiveness of effort estimation: This metric category covers effort estimation accuracy (55%) of Table 3. The previous metric categories do not sufficiently capture this high-frequency (55%) metric, and we separate this single-item category from the rest.

Developers want a dashboard to be able to notify them when the development process is on the wrong track. However, we have also spotted the difficulty of supporting this notification effectively — it is difficult to picture the right track, when developers are currently having difficulties in estimating the effort necessary to complete a task. For this reason, we argue that research on effort prediction deserves strong attention.

5 VALIDATION OF GQM AND DASHBOARD

5.1 Demographics

As a result of the interviews, responses from 30 different companies were collected. Most of respondents (66%) have a Bachelor's degree,

Table 1: Metrics for software development process

Metric	Frequency
Iteration burndown charts	81%
Team velocity	51%
Number of detected defects	39%
Number of unresolved defects	36%
Percentage of passed tests	31%
Release burndown charts	28%
Code coverage	24%
Cumulative flow diagram	13%
Lead time	13%
Average cycle time	10%

Table 2: Metrics for software quality

Metric	Frequency
Code coverage	67%
Defect density	49%
Coupling	16%
Method/class length	10%
Lack of Cohesion of Methods (LCOM)	4%

Table 3: Metrics for process quality

Metric	Frequency
Effort estimation accuracy	55%
Schedule variance	39%
Amount of completed tasks per unit of time	34%
Defect removal efficiency	25%
System spoilage	19%
Percentage of features cut during the project	18%

27% hold Master's degree, and a 5% are PhDs. Work experience differs from 1 to 30 years with the median 5 years. The majority are men (86%), 14% are women. Fifty five percent of respondents work in micro-companies (less than 10 employees), 24% in large companies (more than 500 employees), 14% in medium-sized companies (250 to 500 employees), and 11% in small companies employees (from 10 to 50). Percentage of respondents that apply a methodology based on Agile principles is 65%, while 19% of respondents work on Scrum methodology, and only 5% use Waterfall methodology; 11% use other methodologies. Almost a half (55%) of respondents work in a team of less than 5, 36% in a team of 6 to 10 people, and 9% in a team of 11 to 15 people. Since the respondent can combine posts, the distribution of the roles of the respondents is as follows: 88% of respondents are developers, 22% analysts, 19% technical leaders, and 13% one of the scrum-master, manager or tester.

5.2 "Typical" GQM validation

In the first part of the survey we were checking the relationship between metrics and goals. Validation of goal **G2** did not give enough results for analysis due to respondents' lack of awareness of the company's budget. Therefore, we do not consider the goal **G2** in this section.

For the goal **G1** 78% of respondents recognized the problem with the assessment of efforts. The key metric for 90% of respondents was the "Iteration Burndown Chart"; 76% mentioned the "Effort Estimation Accuracy"; 66% focused on the "Team Velocity Chart". Among those who recognized the problem 79% confirmed that they experienced a similar situation in the last 5 iterations. This confirms the importance of the estimation of effort in the GQM and in the corresponding dashboard. Relevant metrics for this goal are based on the "Iteration Burndown", "Effort Estimation Accuracy", and "Team Velocity" (listed in the order of importance).

For **G3** we got the following result. During the interviews, 100% of respondents recognized the problem with the quality of the code. Thus, all respondents confirmed the importance of the goal related to the quality of the code and the corresponding problem. The distribution of a key metric is the following:

- Passed Tests (97%),
- Code Coverage (91%),
- Unresolved Defects (76%),
- Class / Method Length (64%),
- Iteration Burndown Chart (55%),
- Defect Removal (55%),
- Detected Defects (55%) and Defect Density (45%).

Among respondents who recognized the problem 60% confirmed that they had experienced similar situation in last 5 iterations. The "Average lead" time and "Average cycle" time metrics were not taken into account due to the impossibility of their interpretation by the respondents. In the second part of the interview we were checking the availability of metrics in real-world environment. Indeed, validation of the typical GQM depends on the availability of metrics. Respondents were asked about the PMT and about software solutions for quality control of the code. Developers use the following tools for metrics collection: Jira (35%), YouTrack (22%), Trello (19%), Team Foundation Server (14%); and 10% of respondents were using other software products. For code quality tracking respondents use Sonar (57%), Coverage.py (11%), different products (21%); and 11% of respondents use no tools.

5.3 Dashboard validation

In the first part of the interview, the respondents interacted with the prototype of the dashboard, see Figure 2. It displayed 13 metrics: Iteration burndown chart, Team velocity, Release burndown chart, Average cycle time, Average lead time, Code coverage, Passed tests, Defect removal efficiency, Defected density, Class / method length, Effort estimation accuracy. Each metric was placed on a separate tile. After interaction with the dashboard, respondent were asked the question: "What would you like to add to the functionality of the dashboard and why?". The answers were as follows:

- history and forecasted change in the metric (71%),
- ability to view metrics for an individual team member (43%)
- possibility of filtering and aggregating metrics (37%)
- explanation of possible reasons for changing the metric (24%)
- indication of the number of completed tasks (17%)
- indication of the remaining time of the iteration (10%)
- displaying a hint about actions in the current situation (10%)

The respondents also reported that the function of displaying critical changes, which was implemented in the prototype of the dashboard, was most useful during the decision-making process. Thus, the functionality of the dashboard should have the following objectives:

- view changing metric,
- predict the change in the metric,
- display critical values of metrics,
- filter and aggregate metrics,
- recommend action on the situation,
- display progress of work.

This list of functional is consistent with the first part of the research and complements it.

6 VALIDITY OF RESULTS

In the context and with the intrinsic limitations of an observational study run with an empirical constructivist approach, we claim that it is reasonable to generalize "somehow" the results of this study beyond our study.

Such a claim, indeed, is "loose" since (1) all the work focuses mostly on observational analysis and does not employ any inferential statistics to assess the significance of the results (problem of internal validity), (2) using a questionnaire with closed answers exposes us to the risk of the respondent bias (problem of construct validity), (3) the respondents were self-selected, and no randomization process occurred (problem of external validity) However, we do think that our findings have the potential to be widespread to a wider population, since we took precautions to mitigate the above three classes of threats to validity even if it is not possible to eliminate them. Our work is a first observational study not employing inferential statistics, thus the issue of the significance of the findings is important but not essential. Still, only replications to this study would make our results stronger, and for this purpose we are ready to share our instrument to any interested researcher. In particular, we conducted the validation of the GQM and the precooked dashboard.

7 CONCLUSIONS

In this study, we have interviewed developers from various companies, in an attempt to obtain information necessary to build an effective developer dashboards. We have identified five distinct kinds of metrics: (1) metrics to show the current progress state of the project, (2) metrics to show the speed of work performed, (3) metrics to show the status of testing, (4) metrics to show the status of software quality, and (5) metrics to show the effectiveness of effort estimation. We have observed that, in general, developers are more concerned about monitoring whether the development process is on track than monitoring software quality. Developers want a dashboard to be able to notify them when the development process is on the wrong track. We have connected these metrics with the goals developers want to achieve, through a GQM model. Finally, we have validated the typical GQM using interviews with a separate set of respondents.

ACKNOWLEDGMENT

The authors thank Innopolis University for funding this study.

REFERENCES

[1] Victor R Basili. 1992. Software modeling and measurement: the Goal/Question/-Metric paradigm. (1992).
[2] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. 1994. The Goal Question Metric Approach. In *Encyclopedia of Software Engineering*. Wiley.
[3] Victor R. Basili and David M. Weiss. 1984. A Methodology for Collecting Valid Software Engineering Data. *IEEE Trans. Software Eng.* 10, 6 (1984), 728–738.
[4] Olga Baysal, Reid Holmes, and Michael W. Godfrey. 2013. Developer Dashboards: The Need for Qualitative Analytics. *IEEE Software* 30, 4 (2013), 46–52.
[5] Patrik Berander and Per Jönsson. 2006. A goal question metric based approach for efficient measurement framework definition. In *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*. ACM, 316–325.
[6] Jacob T Biehl, Mary Czerwinski, Greg Smith, and George G Robertson. 2007. FASTDash: a visual dashboard for fostering awareness in software teams. (2007), 1313–1322.
[7] Trevor G Bond and Christine M Fox. 2013. *Applying the Rasch model: Fundamental measurement in the human sciences*. Psychology Press.
[8] Jan Bosch and Helena Olsson. 2017. Towards Evidence-Based Organizations: Learnings From Embedded Systems, Online Games And Internet of Things. *IEEE Software* PP, 99 (2017). Early Access Article.
[9] Eric Bouwers, Arie van Deursen, and Joost Visser. 2013. Software Metrics: Pitfalls and Best Practices. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE '13)*. IEEE Press, Piscataway, NJ, USA, 1491–1492.
[10] Stephan Few. 2006. Information Dashboard Design. (2006).
[11] Adrian Furnham. 1986. Response bias, social desirability and dissimulation. *Personality and individual differences* 7, 3 (1986), 385–400.
[12] Vladimir Ivanov, Alan Rogers, Giancarlo Succi, Jooyong Yi, and Vasilii Zorin. 2017. What Do Software Engineers Care About? Gaps Between Research and Practice. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2017)*. ACM, New York, NY, USA, 890–895. DOI: http://dx.doi.org/10.1145/3106237.3117778
[13] Mikkel R Jakobsen, Roland Fernandez, Mary Czerwinski, Kori Inkpen, Olga Kulyk, and George G Robertson. 2009. WIPDash: Work item and people dashboard for software development teams. (2009), 791–804.
[14] Andrea Janes, Alberto Sillitti, and Giancarlo Succi. 2013. Effective dashboard design. *Cutter IT Journal* 26, 1 (2013).
[15] Andrea Janes and Giancarlo Succi. 2014. *Lean Software Development in Action*. Springer, Heidelberg, Germany. DOI: http://dx.doi.org/10.1007/978-3-642-00503-9
[16] Yasser Khazaal, Mathias van Singer, Anne Chatton, Sophia Achab, Daniele Zullino, Stephane Rothen, Riaz Khan, Joel Billieux, and Gabriel Thorens. 2014. Does Self-Selection Affect Samples'Representativeness in Online Surveys? An Investigation in Online Video Game Research. *Journal of Medical Internet Research* 16, 7 (july 2014), e164.
[17] Walid Maalej, Rebecca Tiarks, Tobias Roehm, and Rainer Koschke. 2014. On the Comprehension of Program Comprehension. *ACM Trans. Softw. Eng. Methodol.* 23, 4, Article 31 (Sept. 2014), 37 pages.
[18] Wilhelm Meding. 2017. Effective monitoring of progress of agile software development teams in modern software companies: an industrial case study. In *Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement*. ACM, 23–32.
[19] Philip M. Podsakoff, Scott B. MacKenzie, Jeong Yeon Lee, and Nathan P. Podsakoff. 2003. Common Method Biases in Behavioral Research: A Critical Review of the Literature and Recommended Remedies. *Journal of Applied Psychology* 88, 5 (10 2003), 879–903.
[20] Paul R. Rosenbaum. 2010. *Design of Observational Studies*. Springer, New York.
[21] Alberto Sillitti, Giancarlo Succi, and Stefano De Panfilis. 2006. Managing non-invasive measurement tools. *Journal of Systems Architecture* 52, 11 (2006), 676–683.
[22] Prashanth Harish Southekal and Ginger Levin. 2011. Validation of a generic GQM based measurement framework for software projects from industry practitioners. In *Cognitive Informatics & Cognitive Computing (ICCI* CC), 2011 10th IEEE International Conference on*. IEEE, 367–372.
[23] StackOverflow. 2017. *Developer Survey Result 2017*. Technical Report. Available online at url: https://insights.stackoverflow.com/survey/2017 and retrieved on August 16th, 2017.
[24] Gergely Szolnoki and Dieter Hoffmann. 2013. Online, face-to-face and telephone surveys – Comparing different sampling methods in wine consumer research. *Wine Economics and Policy* 2, 2 (2013), 57 – 66.
[25] Christoph Treude and Margaret-Anne Storey. 2010. Awareness 2.0: staying aware of projects, developers and tasks using dashboards and feeds. In *ICSE*. 365–374.
[26] David L. Vannette and Jon A. Krosnick. 2014. *Answering Questions: A Comparison of Survey Satisficing and Mindlessness*. John Wiley & Sons, Ltd, 312–327.
[27] Ogan M Yigitbasioglu and Oana Velcu. 2012. A review of dashboards in performance management: Implications for design and research. *International Journal of Accounting Information Systems* 13, 1 (2012), 41–59.