How Practitioners Perceive the Relevance of Software Engineering Research

David Lo
School of Information Systems
Singapore Management University
Singapore
davidlo@smu.edu.sg

Nachiappan Nagappan
Microsoft Research
Redmond, WA
USA
nachin@microsoft.com

Thomas Zimmermann
Microsoft Research
Redmond, WA
USA
tzimmer@microsoft.com

ABSTRACT
The number of software engineering research papers over the last few years has grown significantly. An important question here is: how relevant is software engineering research to practitioners in the field? To address this question, we conducted a survey at Microsoft where we invited 3,000 industry practitioners to rate the relevance of research ideas contained in 571 ICSE, ESEC/FSE and FSE papers that were published over a five year period. We received 17,913 ratings by 512 practitioners who labelled ideas as essential, worthwhile, unimportant, or unwise. The results from the survey suggest that practitioners are positive towards studies done by the software engineering research community: 71% of all ratings were essential or worthwhile. We found no correlation between the citation counts and the relevance scores of the papers. Through a qualitative analysis of free text responses, we identify several reasons why practitioners considered certain research ideas to be unwise. The survey approach described in this paper is lightweight: on average, a participant spent only 22.5 minutes to respond to the survey. At the same time, the results can provide useful insight to conference organizers, authors, and participating practitioners.

Categories and Subject Descriptors
D.2 [Software Engineering]

General Terms
Measurement, Experimentation

Keywords
Software Engineering Research, Survey, Industry

1. INTRODUCTION
The number of published software engineering papers has been growing over the past few years. For example, the number of papers published in ICSE almost doubled in the last 5 years from 50 in 2009 to 99 papers in 2014. Other conferences in software engineering have observed a similar growth. Does this mean that the relevance of software engineering has grown as well?

For any community, it is important to reflect on successes and failures and to assess if it is moving in the right direction. The Impact project [1] by ACM SIGSOFT investigated if and in what areas, software engineering research had an impact on practice. Areas that were identified as part of the project included modern programming languages, software configuration management, and inspections. In addition to impact, other "health" aspects of software engineering research have been analyzed such as the health of conferences [2] or novel peer review models [3]. The question of impact and relevance of software engineering research has been raised by practitioners in industry, funding organizations, and researchers themselves. A complicating issue is that the actual impact of some research in practice is often only known after many years. Researchers typically have to speculate what will have the most impact in the future.

In this paper, we propose a lightweight technique to gather rapid feedback on how practitioners perceive the relevance of software engineering research. The process is as follows: as part of a survey, present short summaries of research papers, which succinctly capture research ideas contained in them, to a panel of practitioners and ask them to rate how important each research idea is for their work: *"In your opinion, how important are the following pieces of research?"* In the survey, participants can rate research as *Essential, Worthwhile, Unimportant, Unwise*, or state *"I Don't Understand"*. To keep the time investment low for practitioners, we keep the summaries short (shorter than an abstract) and limit the number of summaries to be rated by each participant (by selecting a given number of summaries randomly for each participant).

To demonstrate the feasibility of the technique, we invited 3,000 engineers in Microsoft to mark 571 research papers from five years of the ICSE (2010-2014), ESEC/FSE (2009, 2011, 2013) and FSE (2010, 2012) conferences. We received 17,913 ratings from 512 practitioners (response rate 17%). Such data can be used to empirically answer several important questions to guide software engineering research:

- How do (Microsoft) practitioners view software engineering research as a whole?
- What research ideas do (Microsoft) practitioners consider to be most important?
- Why (Microsoft) practitioners view some research ideas as unwise?

While the answers to these three questions may not generalize beyond the context of Microsoft engineers, they highlight the potential of what insight we can obtain if as a community we repeat this experiment for other populations, e.g., with practitioners from multiple companies.

To be more specific, the answer to the first question could serve as a *health indicator*, e.g., is software engineering research relevant and does it remain relevant. The answer to the second question could help researchers to *prioritize their research efforts*. And the answer to the third question could help researchers *avoid pitfalls* that can make research less appealing to practitioners.

To be very upfront the goal of our study is to NOT rank papers based on relevance. We do not intend or envision this paper to play a role in ranking papers, which we think is contrary to the spirit of fundamental research. Our goal is to provide a new perspective on how to assess the perceived relevance of today's software engineering research as viewed by practitioners.

This paper makes the following contributions:

1. We propose a survey-based framework to assess the relevance of software engineering research by involving practitioners.
2. We present findings from a study with engineers at Microsoft and point out opportunities on how to improve it with a community-wide effort.
3. We identify reasons why practitioners perceive certain high-level research ideas as "Unwise".

The remainder of this paper is structured as follows. First, we describe the experimental design of our study in Section 2. We then describe the results of our study in Section 3. In Section 4, we discuss the implications of our results as well as limitations of our findings. In Section 5, we describe related work. In Section 6, we conclude the paper and describe our future work plans.

2. EXPERIMENTAL DESIGN

To help assess the relevancy of software engineering research, we propose a framework which includes the following steps:

1. *Select and summarize the papers of interest.*
   There are many possible ways that papers can be selected, for example based on conferences (e.g., ICSE, ESEC/FSE, FSE), based on topic (e.g., testing, program analysis), or based by year.

   The summarization step reduces each paper into a short text summary of limited size. It is needed so that practitioners can understand the paper well enough to provide feedback, without the need to spend much time to understand the technicalities. Rather, they should be able to focus on how the research idea can help their day-to-day activities.

2. *Select the participants.*
   It is important to select a representative set of participants for the survey. Practitioners typically have different needs depending on their roles (developers, testers, etc.) and it is important to capture the viewpoints of a diverse set of practitioners.

   The number of participants to invite depends on several factors: Assuming we have P papers to rate, and we want each paper to receive R ratings on average, we require P x R ratings. Each participant can rate K papers and the expected response rate to a survey is S. Then we need to invite (P x R) / (K x S) practitioners to the survey.

3. *Run the survey and collect feedback.*
   The following information can be collected as part of a survey: demographics, the ratings of the papers, as well as additional feedback to follow up on previous responses.

4. *Analyze the data.*
   The responses can be analyzed using a set of metrics. For free form answers, qualitative techniques such as open card sort can be used.

Several of the above decisions (summaries instead of abstract, a random paper selection for each participant) were made with the goal to *keep the time investment low* for practitioners.

2.1 Paper Selection and Summarization

We selected full research track papers published in the ICSE, ESEC/FSE, and FSE conferences during a time period of 5 years. This includes 184 papers from the meetings of ESEC/FSE and FSE (2009-2013), and 387 papers from ICSE (2010-2014), for a total of 571 papers. We believe that this is a representative sample of software engineering research as ICSE, ESEC/FSE, and FSE are general conferences as opposed to specialized conferences such as ISSTA for testing to ICSME for maintenance and evolution. We did not include journal publications because they often extend previous conference papers.

As participants would not have the time to read long abstracts or the entire paper itself, we created for each paper a short descriptive summary that contains the key ideas of the paper. The first author read the abstract of each paper and constructed a summary (a few sentences) to capture the gist of the paper. If the abstract was unclear, the first author also downloaded and read the paper.

To improve the paper summaries, the second author verified the quality of the initial summary created by the first author and provided suggestions for improvement. After these suggestions were incorporated into an updated summary, we piloted the summaries to a small set of practitioners to get their feedback as suggested by Kitchenham and Pfleeger [4]. We further improved our summaries based on the practitioners' feedback.

2.2 Participant Selection

We selected full time employees of Microsoft as participants whose job roles included development, test, and program management. For the selection, we followed Kitchenham and Pfleeger's [4] advice on the need to understand whether the respondents had enough knowledge to answer the questions in an appropriate manner. For this, we restricted the people invited to participate in the survey to people in technical roles (no sales or marketing employees).

With P=571 papers, R=30 ratings per paper, K=40 ratings per participant, and response rate of S=0.15 (estimated based on response rates from previous surveys), we estimated the number of practitioners that we need to invite to be 2,855, which we rounded up to 3,000. We then randomly picked 3,000 full time employees from the Microsoft employee database who were working in technical roles. Since Microsoft has more employees working in development roles, the random selection naturally has a higher proportion of participants working in development roles than participants working in the test and program management roles.

Respondents were anonymous, but as a thank you for the participants' time, they could enter their name (separate from their survey responses) into a raffle of three $75 Amazon gift certificates at the end of the survey.

2.3 Feedback Elicitation

In order to elicit feedback from a wide range of participants in a scalable way, we used an online survey. We designed the survey such that participants required as little effort as possible to complete it, e.g., it was self-contained and included all relevant information. We limited the response types to numerical, Likert-scale, and short free form answers as suggested by Kitchenham and Pfleeger [4]. The survey we used within Microsoft is shown in Figure 1 (see our technical report [5] for the full survey).

We piloted the survey with a small set of practitioners to get their feedback and improve the survey.

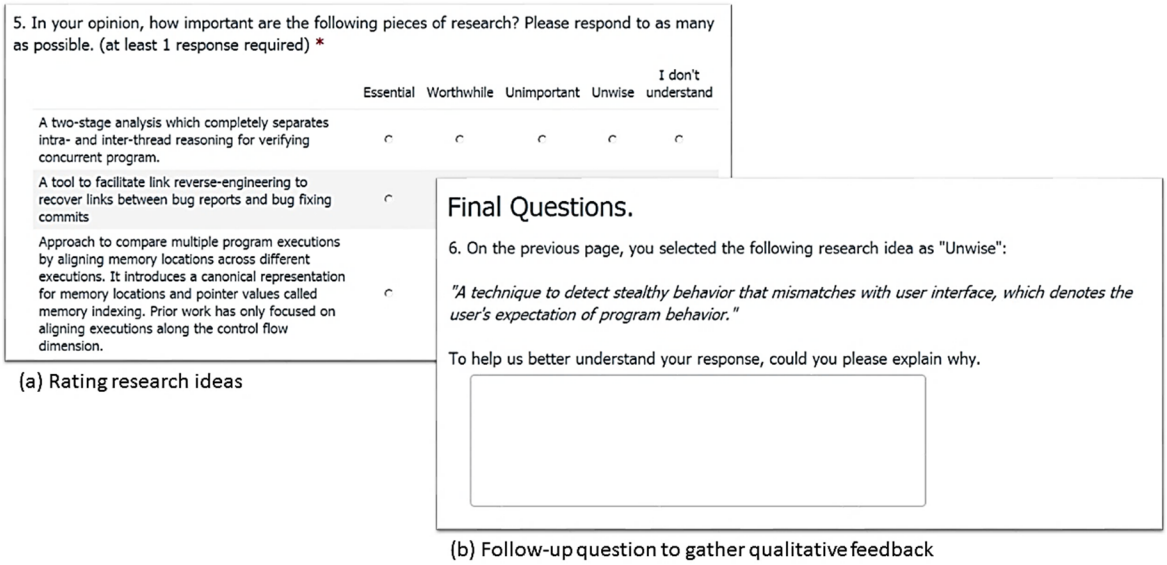


Figure 1. Some questions in our survey. On the first page of the survey, we asked for demographics and rating of research ideas (Figure 1.a). On the second page, we asked a follow-up question to gather additional qualitative free text feedback, when participants rated a research idea as "Unwise" (Figure 1.b)

We captured the following information as part of the survey:

Demographics. (answers were required):

- *Primary work area:* Development, Test, Program Manager, Other
- *Role:* Individual contributor, Lead, Architect, Manager, Executive, Others
- *Experience in years* (decimal value)
- *Major in Computer Science* (Boolean value)
- *Has advanced (i.e., postgraduate) degree* (Boolean value)

Collecting some basic information about the participants allowed us to break down the results by groups, e.g., developers, testers, etc.

Ratings of research ideas (see Figure 1.a). For each participant, we randomly selected 40 papers from the collection of 571 papers. At least one paper had to be rated to complete the survey.

We then present summaries of those papers to the participants and ask them to rate how important each paper is for their work: *"In your opinion, how important are the following pieces of research?"* Following the rating categories used by Begel and Zimmermann [6], participants can label a research idea as: "Essential", "Worthwhile", "Unimportant", "Unwise", and "I Don't Understand". The last category was included to address the diverse background of participants—not all participants will understand all technologies.

We chose to ask the question "In your opinion, how important are the following pieces of research?" to allow practitioners to provide feedback based on their personal experience. We decided *not* to ask about the willingness to adopt because other research has shown that adoption depends on many different factors (social, cultural, and educational factors, exposure, and many more [7] [8]), which are often external to the actual research. A reliable assessment of the adoptability would require a significant time commitment by practitioners. Lastly, adoption also heavily depends on the type of research, e.g., tools and techniques are adopted differently than say empirical studies.

Follow-up: Rational behind specific ratings (see Figure 1.b). One limitation of ratings is their inability to capture reasons why participants viewed a certain research idea in a particular way. To capture the reasons behind some ratings, we included follow-up questions.

More specifically, if there was one or more summaries that a participant labeled as "Unwise", we randomly selected one of those summaries and asked participants to elaborate the reasons why they felt that the high-level research idea was unwise to pursue. Participants could enter free text to express their thoughts; the follow-up question was optional.

2.4 Data Analysis

We compute several statistics to characterize the overall perspectives that practitioners have on software engineering research. We measure the proportion of ratings that are Essential (best response), Essential or Important (positive feedback), or Unwise (worst response), respectively. More formally, let *E, W, Ui,* and *Uw* denote the number of essential, worthwhile, unimportant, and unwise ratings received.

- **E-score:** The percentage of ratings that are "Essential"

$$\text{E-score} = \frac{E}{E + W + Ui + Uw}$$

- **EW-score:** The percentage of ratings that are "Essential" or "Worthwhile".

$$\text{EW-score} = \frac{E + W}{E + W + Ui + Uw}$$

- **U-Score:** The percentage of ratings that are "Unwise"

$$\text{U-score} = \frac{Uw}{E + W + Ui + Uw}$$

The statistics can be computed for different groups, e.g., all ratings, ratings by certain demographics, ratings for specific conferences, or ratings for individual papers.

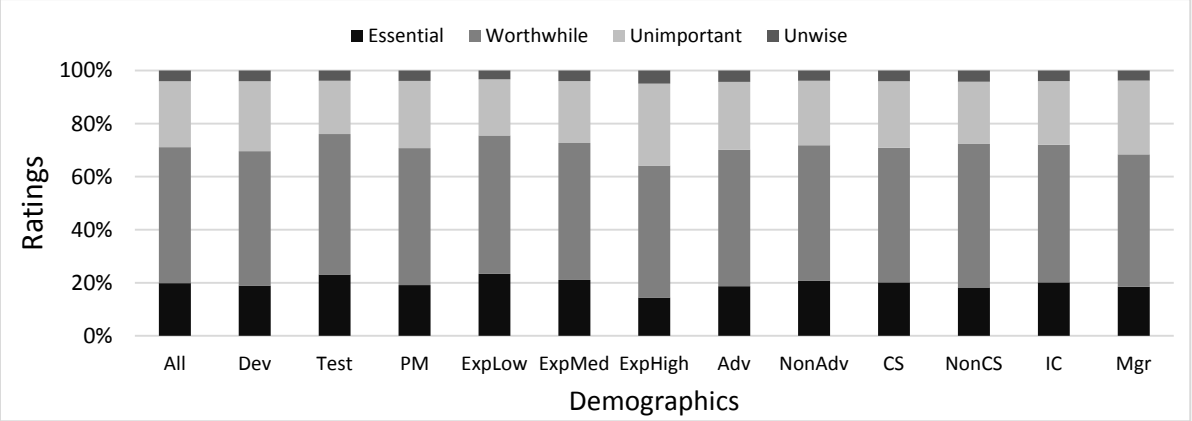


Figure 2. Percentage of ratings of various demographics.

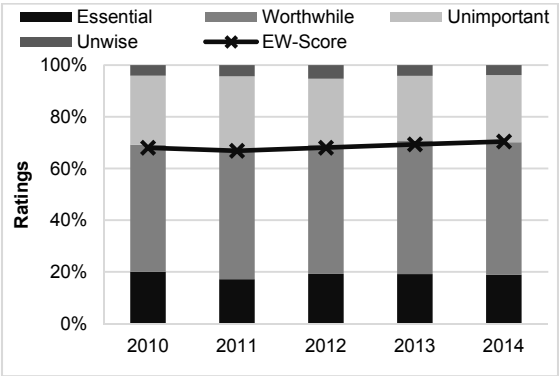


Figure 3. Relevancy scores from 2010-2014 (ICSE)

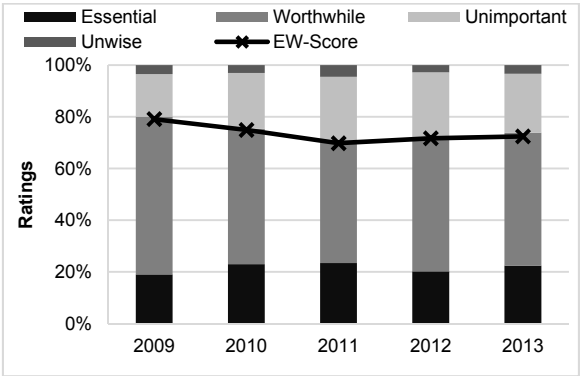


Figure 4. Relevancy scores from 2009-2013 (ESEC/FSE, FSE)

To group the reasons why research ideas were selected as unwise, we use an open card sort [9]. Card sorting is widely used to create mental models and derive taxonomies from data. Our card sort consisted of two phases: in the *preparation* phase, we create one card for each response to the follow-up question to why a research idea is unwise. In the *execution* phase, cards are sorted into meaningful groups with a descriptive title. Our card sort was *open*, meaning we had no predefined groups; instead, we let the groups emerge and evolve during the sorting process. By contrast, a closed card sort has predefined groups, which is typically used when the themes are known in advance. All three authors jointly sorted the cards.

3. RESULTS

We invited 3,000 randomly selected practitioners who were working in a technical role to take the survey; 512 participants respond (response rate 17%). Among the participants, 291 (56.8%) of them work as developers, 87 (17.0%) as testers, and 102 (19.9%) as program manager. The participants collectively provided 17,913 ratings. Each paper was rated by 16 to 47 participants (since we randomly picked papers to show to survey respondents the number of ratings per papers follows a hypergeometric distribution). 217 participants rated at least one paper as "Unwise", 173 of them (79.7%) provide their reasons for a randomly selected paper. Of the ratings, 2,745 (15.3%) were "I Don't Understand", which demonstrates the need of having a way for participants to not provide a rating if they don't have enough knowledge or context to assess a research idea.

To show the potential of practitioner feedback, we consider three questions:

1. How relevant is software engineering research in the practitioner's perspectives? (Section 3.1)
2. What are highly rated research topics that practitioners deem essential? (Section 3.2)
3. What are the reasons why practitioners consider certain research topics to be unwise? (Section 3.3)

3.1 Relevance of Software Engineering Research

Figure 2 shows the percentage of ratings across the various categories for different *demographics*:

- all participants (All),
- participants that are developers (Dev),
- participants that are testers (Test),
- participants that are program managers (PM),
- participants with low experience, which we define as the 25% with the least experience in years (ExpLow)
- participants with medium experience (ExpMed),
- participants with most experience, which we define as the 25% with the most experience in years (ExpHigh),
- participants with advanced degree (Adv),
- participants without advanced degree (NonAdv),
- participants with CS degree (CS),
- participants without CS degree (NonCS),
- participants who are individual contributors (IC), and
- participants who are managers (Mgr).

Table 1. Highly rated research ideas. Total is the number of people who rated a paper and did not select "I Don't Understand"

	Paper Summary	Total	E-Score	EW-Score	U-Score
P1	An approach to help developers identify and resolve conflicts early during collaborative software development, before those conflicts become severe and before relevant changes fade away in the developers' memories.	39	0.62	0.85	0.00
P2	Technique that clusters call stack traces to help performance analysts effectively discover highly impactful performance bugs (e.g., bugs impacting many users with long response delay).	30	0.60	1.00	0.00
P3	Symbolic analysis algorithm for buffer overflow detection that scale to millions of lines of code (MLOC) and can effectively handle loops and complex program structures.	29	0.55	0.97	0.03
P4	Automatic generation of efficient multithreaded random tests that effectively trigger concurrency bugs.	29	0.55	0.90	0.03
P5	Debugging tool that uses objects as key abstractions to support debugging operations. Instead of setting breakpoints that refer to source code, one sets breakpoints with reference to a particular object.	29	0.55	0.90	0.03
P6	Technique to make runtime reconfiguration of distributed systems in response to changing environments and evolving requirements safe and being done in a low-disruptive way through the concept of version consistency of distributed transactions.	31	0.55	1.00	0.00
P7	A new technique that not only detects leaks, but also points developers to the locations where the underlying errors may be fixed.	25	0.52	1.00	0.00
P8	An approach which automatically prevents database deadlocks which happen when a database is being accessed by multiple database-intensive applications which hold locks to different tables.	34	0.50	0.94	0.00
P9	A technique to engineer applications with a self-healing layer for service-oriented systems that dynamically reveals and fixes interoperability problems.	30	0.50	0.93	0.03
P10	A new algorithm for Bayesian inference over probabilistic programs, based on data flow analysis techniques from the program analysis community to deal with loops. Inference refers to the process of calculating the posterior distribution specified by the probabilistic program.	14	0.50	0.79	0.00

From the figure we can observe that all demographics give more "Essential" and "Worthwhile" ratings as compared to "Unimportant" or "Unwise". The EW-score for all participants is 71%, and the E-score is 20%. This is an encouraging finding as it shows that practitioners do value work done in the software engineering research community. However, it also shows that there is room for improvement.

We observed several differences between the demographics (all statistically significant at a p-level of 0.001):

- Testers were more positive about the relevance of software engineering research (EW-score of 76%, E-score of 23%) than developers or program managers.
- As experience increased, participants were more critical and considered more studies as unimportant as well as less studies as essential. For the participants with low experience (ExpLow), the EW-score is 75% and the E-score is 23%; in contrast, for participants with high experience (ExpHigh), the EW-score drops to 64% and the E-score to 14%.
- Participants who were in Management roles (Mgr) were more critical than individual contributors (IC). They had an EW-score of 68% and E-Score of 18%.

We did not observe any statistically significant difference in ratings between participants with and without advanced degree. Similarly, there was no statistically significant difference between participants with and without CS degree.

In addition to demographics, we can break down relevance scores by *conference* and *year*. This can help organizers to ensure that the conference program matters for practitioners. Figures 3 and 4 show the changes in the relevance scores for papers published in ICSE, ESEC/FSE, and FSE in the last 5 years. Over time, the EW-scores are close to or above 70%, which shows that the practitioners who we surveyed found that the majority of software engineering research presented at ICSE, ESEC/FSE, and FSE worthwhile to be pursued.

We also notice that except for a drop for ESEC/FSE and FSE from 2009 (EW-score of 79%) to 2011 (EW-score of 70%), the scores for ICSE, ESEC/FSE, and FSE are relatively stable. This shows the health of the conference series in general is good along the years. It would be great if their health could improve further in the years ahead.

Monitoring health of conferences is one example of how the approach in this paper could be used. A conference could poll a representative set of practitioners (or attendees for that matter) every year to see if relevance scores of the conference are improving and to ensure that the presented work matters for practitioners.

3.2 Highly Rated Research Ideas

Table 1 highlights the ten paper summaries rated the highest by the 512 participants in our survey. The papers are sorted in terms of their E-score scores (descending), followed by EW-scores (descending), followed by U-scores (ascending). Among the top-10 summaries, at least half of the respondents agreed that the work is essential. These papers cover a diverse set of topics including improving system performance, debugging tools, adaptive systems, testing multithreaded programs, and collaboration conflict detection, among others. Some readers may be surprised with summary P10 about a new algorithm for Bayesian inference over probabilistic programs. This idea had a high number participants choosing "I don't understand", which suggests that it may be a specialized topic that is highly relevant to some participants but not to the ones who are unfamiliar with the topic.

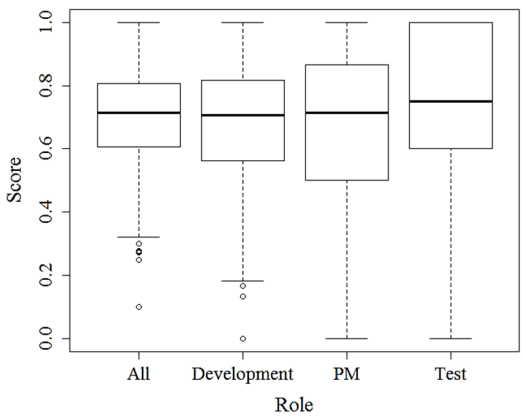


Figure 5. Distribution of EW-scores per paper

In addition, we ranked summaries by the job role (developer, tester, and program manager). We required a paper summary to be rated by at least five people to be included in a role-specific ranking. We show the list of top-5 papers for each role in our technical report [5]. We made the following observations:

- Developers are highly interested on improving system performance, detecting collaboration conflicts, debugging techniques, and detecting concurrency bugs.
- Testers are highly interested in system monitoring, adaptive systems, finding linkages between bug reports to the fixing commits, and lightweight verification tools.
- Program managers are highly interested in a diverse set of topics including building agile teams, team awareness, software product line construction, bug finding, and debugging tools, among others.

Figure 5 shows a boxplot of the distribution of EW-scores computed on a per paper basis. We can observe that practitioners rate most papers favorably. For the majority of papers the EW-score is higher than 0.5. Again we can observe that testers are more positive than developers and program managers.

These findings can direct research into areas that are considered to be relevant by practitioners.

3.3 Reasons Why (Certain) Research is Considered "Unwise"

In a few cases (3.4% of all votes), practitioners rated research ideas as "Unwise". In those cases, we asked a follow-up question for why respondents think the research idea is unwise.

To identify common reservations, we took the responses and clustered them into groups. We performed an open card sort [9] to create the groups. First we printed each response on a card, we then discussed the comments and iteratively sorted them into groups. Our card sort identified eight categories that are discussed below. We ignored cards that contained no rationale (*"Doesn't sound wise to me"*) or when participants rejected an idea solely because it was not using Microsoft technology (*"I think we should focus on Windows First"*).

A tool is not needed. This group consists of comments where the practitioner perceived a tool as not useful to their daily work, e.g., the tool cannot help make their tasks being performed easier or with higher quality. The respondent deemed the current state-of-practice good enough and believed that no additional support is necessary.

- *"The tool that would result would not be something I would use or can imagine anyone else using"*
- *"I don't know how it could be used for daily work"*
- *"I don't believe that a framework will make the design and maintenance of such systems any easier",*
- *"The proposed tool is already available in the form of TFS or SharePoint list"*

An empirical study is not actionable. The practitioners perceived that the subject analyzed by an empirical study was not relevant and/or the findings of the study were deemed to be not actionable or of little benefit.

- *"I wouldn't expect anything actionable or relevant to come out of this study"*
- *"I don't see what's the value to study the difference between these two development (methodologies)"*
- *"Don't see any need for this study since enough is known about common fallacies of this type",*
- *"Don't know why there would be any benefit of knowing the answer to the proposed question", etc.*

Generalizability issue. Practitioners criticized that a study was limited to a few systems and findings found by analyzing these systems might not be applicable to real systems that matter to them.

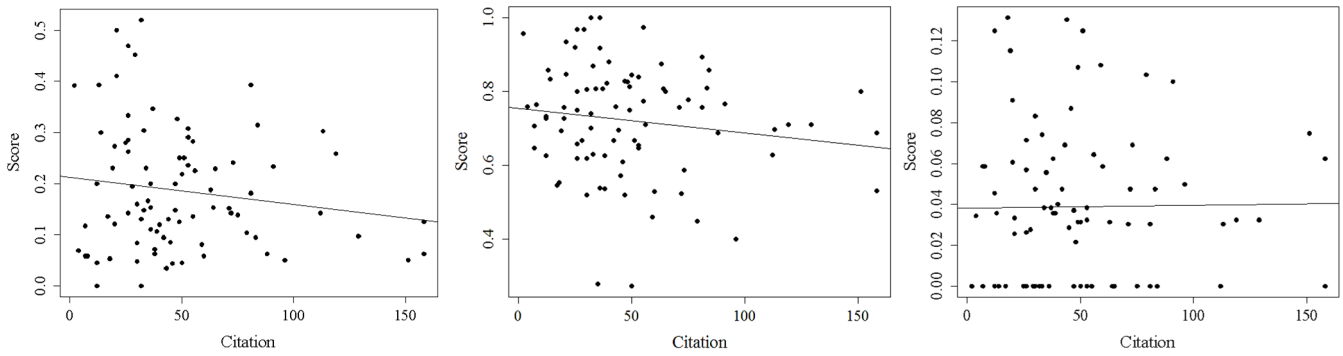
- *"Empirical study on this platforms may not be reusable on others"*
- *"Case study for a project is always less useful than researching around a topic. Lessons learned from one project can be very specific to this project"*
- *"Might want to consider bugs in same applications over different platforms"*
- *"Developers are not alike"*

A subcategory of generalizability was scalability. Practitioners deemed a technique to be unlikely to cope with the size, complexity, and variability that characterize systems that practitioners are working on.

- *"I don't see this being used for large-scale systems"*
- *"The set of software update that needs testing is not a small number and new software updates happen almost every week. And it is not the same set of software installed by different users"*
- *"Energy consumption characteristics will vary from device to device and over time"*
- *"From past attempts: - too many states*
  – too many false positives"
- *"For a complex program, there will be too much info, and the developer will not be able to understand"*
- *"As the complexity of the bug goes up, the solution may or may not go up"*

Cost outweighs benefit. Practitioners deemed the cost of using and maintaining a particular tool to be higher than the benefit gained by using the tool.

- *"Development cost of this approach will overkill the gain it gives"*
- *"Huge time investment for little return"*
- *"I believe the cost of implementing and maintain such a solution would be greater than the cost of developers fixing bugs manually"*



(a) E-score (a) EW-score (c) U-score

Figure 6. Citation count vs. relevancy scores (ESEC/FSE 2009 and ICSE 2010)

- *"I have experience with similar systems and I've never seen one where I thought they were of net-value"*

Questionable assumptions about inputs or conditions. Practitioners deemed a particular input/condition crucial towards the success of a research might not hold in practice.

- *"The whole research assumes that there are requirement documents and design documents in software development… which is false in most software projects nowadays"*
- *"Such a tool makes it easier for people to focus on test coverage & state coverage. Which doesn't really help detect bugs"*
- *"Description is often not filled correctly. hence it is unwise to rely on it"*
- *"Analyzing documentation written by humans seems inherently risky. Engineers are not known for writing good documentation, and I suspect that will only get worse as we accelerate our deliverables"*

Disbelief in a particular technology or methodology. Some practitioners had strong disbelief in a particular technology or methodology on top of which a research work is based.

- *"I don't believe in design patterns, force fitting something into a pattern is not wise"*
- *"UML is half dead!"*
- *"Multi-threading is to be avoided at all cost"*
- *"I don't think UML is a good tool to use in the development process"*

Another solution/problem seems better/more important. Practitioners believed that it is better to work on an alternative solution/problem that will also solve the need addressed by some research work.

- *"Not sure if this is the best or the easiest way to find new uses. Usually I look at forums/books/tools for that"*
- *"Making yet another language isn't really solving anything. Instead, give me more functionality within my language and/or give me tools to do these types of things"*
- *"Better organization of how Linux is packaged and distributed would solve this issue without the need of deep analysis and investigations"*
- *"I don't think natural language is that important. Instead helping users find the keywords or tags is should be the focus"*

Proposed solution has side effects. Practitioners perceived that although a research work can solve a problem, it might cause other problems as side effects.

- *"It seems that there could be potentially disastrous results if the automation does not fix the issue it detects correctly. It could induce laziness or uncaring attitude of developers, e.g., 'it doesn't matter if we introduce bugs, the repair app will fix them'"*
- *"Design Patterns … derive their flexibility at the expense of comprehensibility of the interacting parts of a system"*
- *"This approach can introduce "false" exception / bug conditions, increasing the cost and time to market, while introducing the opportunity to over engineer the code"*
- *"Specific techniques to rank devs can lead to devs not working together and lower productivity/morale"*
- *"Drag and drop solutions have always seemed to me as a quick and easy way to write inefficient code"*
- *"It can easily lead to group think with our competition. We need to think outside the box, not just copy"*

4. DISCUSSION

We discuss several aspects of this work: the relationship between citation counts and perceived relevance (Section 4.1), the cost of practitioner feedback (Section 4.2), how feedback could be integrated into conferences (Section 4.3), and limitations of this work (Section 4.4).

4.1 Citation Count vs. Perceived Relevance

We analyzed the relationship between citation counts and the practitioner's perspectives as captured by E-, EW-, and U-scores. Citation counts are often used in academia as a measure of importance and impact. For the analysis, we used the citation counts from ESEC/FSE 2009 and ICSE 2010. We collected the citation counts from Google Scholar on Aug 18, 2014. We ignored more recent instances of the conferences because citations need time to be built up. It would be unfair to compare papers from ICSE 2014 with papers from ICSE 2010 as authors had four more years to read and cite work from ICSE 2010.

Figure 6 shows scatter plots between citations received by papers published in ESEC/FSE 2009 and ICSE 2010 with their relevance scores computed from practitioners' ratings. From the regression lines, we find that the number of citations is not necessarily positively correlated with the three relevance scores. We also computed Spearman correlations. The correlation between the citation count

and E-score is –0.07 with a p-value of 0.53 (not significant); the correlation between the citation count and EW-score is –0.13 with a p-value of 0.23 (not-significant); the correlation between the citation count and U-score is 0.04 with a p-value of 0.73 (again not significant). Thus, we can conclude that there is no correlation between the citation count and relevance scores of papers.

Some papers were cited only a few times, but practitioners considered the work to be solving a problem that is essential to their needs. To illustrate such cases, Table 2 shows the top-5 studies that were favored by practitioners but less so by academia and vice versa. To get this list, we divided the rank of a paper based on citation counts with the rank based on E-scores. Papers favored by academia have high citation counts and relatively low E-scores and for papers favored by practitioners vice versa.

The absence of a correlation shows that the relevance scores introduced in this paper add extra information to the assessment of research that cannot be captured by citation counts.

4.2 Fast, Lightweight, and Inexpensive Assessment of Perceived Relevance

We want to stress that the survey-based approach presented in this paper is a *fast, lightweight,* and *inexpensive* way to assess perceived relevance of research ideas and to collect feedback.

Most participants of our survey responded within the first 3 days. We closed the survey after 8 days and collected more than 17,913 ratings. This process is faster than a typical conference review process that can take 1-3 months.

Surveys with Likert-scales are a lightweight way to collect feedback. The survey tool used for this study (SurveyGizmo.com) provides optimized views for mobile devices which gives participants the freedom to take the survey at any time they like, for example, while waiting in line for lunch. The Fatigue and the Accessibility scores [10] of the survey were estimated to be low by the survey tool.

The cost of running the survey is also low: Summarizing the papers and implementing the survey took approximately 80 hours. The 512 survey participants took 22.5 minutes on average[1] to complete the survey (for a total of 192 hours). Setting up the analysis framework for the survey took another 40 hours. The monetary cost were a license of the survey tool (Enterprise Plan, 1 month) for $199 and 3 Amazon gift certificates as incentive to participate in the survey (each $75; total $225).

We would like to emphasize that survey-based practitioner feedback as proposed in this paper is different and by no means meant to replace the work of program committees (PCs). In addition to checking papers for relevance, PC members must check papers for many other criteria such as originality, presentation, correctness, etc. This requires reading the full paper, a larger time commitment, and often travel to a physical PC meeting to discuss papers with other experts in person.

4.3 Towards Feedback-Driven Conferences

We believe that embedding practitioner feedback into conferences (and maybe even journals) can provide great value to the software engineering community.

[1] The survey tool records start and end times of people taking the survey, the average time to complete the survey was 22.5 minutes. (When computing the average we ignored durations longer than two hours because participants likely got interrupted and completed the survey at a later point in time.)

Table 2. Top studies favored by academia and practitioners

FAVORED BY PRACTITIONERS
A new technique that not only detects leaks, but also points developers to the locations where the underlying errors may be fixed.
A technique to engineer applications with a self-healing layer for service-oriented systems that dynamically reveals and fixes interoperability problems.
A technique to monitor if a system fulfils its requirements expressed as probabilistic properties (e.g., performance, reliability, safety, and availability requirements) at runtime
Automatically detecting security vulnerabilities in client-side self-contained components that interact with one another.
Failure to release unneeded system resources results in resource leaks, which can lead to performance degradation and system crashes. The paper presents a new tool that performs static analysis to find code that causes resource leaks in Java programs.
FAVORED BY ACADEMIA
Empirical study on whether the bug fixes recorded in these historical datasets is a fair representation of the full population of bug fixes.
Technique to verify the correctness of a family of programs in a software product line against a set of properties.
Empirical study of using software defect data from one project to predict defects in another project.
A graph model based on Markov chains, which captures bug tossing history, to better assign developers to bug reports
Over 30 years ago, the preprocessor cpp was developed to extend the programming language C by lightweight metaprogramming capabilities. The paper describes a study that analyzes 40 C projects to investigate how cpp preprocessor is employed to implement variability.

For example, this could work as follows:

1. The Program Committee (PC) reviews the submitted papers and selects the accepted papers.
2. After notification, the Authors of each paper provide a short summary that is used in the survey. While this is extra work, in return the authors would get feedback from practitioners.

   An alternative is to ask the PC members to provide a summary and have the Authors validate the summary.
3. The summaries are then used to for the survey that is sent to the Practitioners.

   To increase the representativeness, the survey should be sent to multiple companies, or even better, a representative panel of industry practitioners (covering different companies and parts of the software industry) who are willing to regularly provide feedback on software engineering research.

   We recommend using a similar scale as in our survey. Instead of following only for research ideas that were rated as Unwise, we suggest to have a feedback option for any research idea. In addition, the survey should provide a way for the Practitioner to trace back a research idea to the actual paper after they have completed the survey.

Such a survey design has advantages for many stakeholders:

- ***Conference organizers*** can use the practitioner feedback to assess the perceived relevance of their conference for industry. They can monitor the scores over time (as illustrated in Section 3.1 for ICSE and FSE) and take steps to increase the relevance, driven by actual data. The survey can also serve as publicity for

the conference and possibly attract extra attendees if done before the conference.

- ***Authors*** can take the (text-based) feedback to improve their research and make it more relevant to practitioners if they want. They also get additional visibility for their work.
- ***Practitioners*** get an overview of the latest research, which several participants appreciated in our study, e.g., *"Thanks for that summary, it is actually interesting by itself"*, or *"Reading through just the titles was a fascinating read – some really interesting work going on!"*

We believe that the approach introduced in this paper is an effective means to help reduce the gap between practitioner needs and software engineering research efforts. Lastly feedback does not have to be limited to practitioners. A conference could survey its attendees to get a sense how happy they were with the paper selection.

4.4 Limitations

We acknowledge the following limitations of findings presented in this paper. The limitations could easily be addressed through a community-driven move to feedback-driven conferences.

The statistics reported in this work depend on the summaries that were created by ourselves. We have followed a process to help improve the quality of the summaries that we generated. It is possible that some participants had poor understanding of some of the summaries. To reduce the impact of this issue, we included an "I Don't Understand" option in the survey and ignored responses marked as such. Note summaries are needed because it is not practical to ask survey participants to read entire papers and many abstracts are not concise enough. Ideally summaries would be created by the authors of the papers and/or PC members as outlined in the previous section.

The findings in this paper are based on practitioner feedback from one company. We acknowledge that perspectives of practitioners in other companies and/or industries such as automotive, aerospace, or banking may be different. As we discussed in Section 4.3, ideally the survey would be send to a representative panel of practitioners. Even though the statistics and insights in this paper come only from one company, we believe that they are still useful because we surveyed a large number of practitioners (more than 500) with diverse backgrounds. While some projects are larger in size at Microsoft, most development practices in the company are adapted from the general software engineering community and also used outside Microsoft. Microsoft is a large organization that produces a wide range of software and hardware products such as operating systems, productivity software, web browsers, video games, search engines, game consoles, tablets, phones, and many more. Technical employees at Microsoft come from many different schools, countries, with many different cultural backgrounds and we argue that they are thus highly representative of developers all over the world [11].

In this work, we focused on assessing research work's *perceived relevancy* in the eyes of Microsoft engineers. Perceived relevancy does not mean that a research work will be adopted by practitioners. In the survey we did not ask developers to answer whether they are willing to adopt a research idea. Asking about adoption assumes the availability of tools, which is not always the case, e.g., for empirical papers. In addition, adoption typically depends on different factors, e.g., social, culture, education, exposure, and many more [7] [8], which often are external to the actual research. These issues make it difficult for developers to provide objective assessments of the adoptability. In reality, an actual decision whether a research contribution can be adopted would require a significant time commitment by practitioners. With this work and the question about relevance, we wanted to explore lightweight feedback techniques.

5. RELATED WORK

Related work falls into three areas: papers related to the SIGSOFT Impact Project, retrospective studies, and attempt to rank the software engineering community.

ACM SIGSOFT Impact Project. Our work is partly inspired by the Impact project performed by ACM SIGSOFT. The goal of the project is to assess the importance of software engineering research among the practitioners. This is done by "a series of studies and briefings, each involving literature searchers and, where possible, personal interviews" [1]. Several research studies under the Impact project has resulted in a number of publications including:

- Ryder and Soffa investigated how exception handling is used today and traced back current state-of-the-practice to studies in software engineering that helped shape the current state-of-the-practice [12].
- Ryder et al. analyzed modern programming languages and documented past studies in software engineering and programming languages that have impact on features in these modern programming languages [13].
- Estublier et al. investigated how software configuration management systems have evolved along the years and the impact of research performed in universities and industries [14].
- Clarke and Rosenblum reported the historical development of runtime assertion checking and described how it has been used in some industries as reported in a number of publications [15].
- Emmerich et al. investigated a number of successful middleware technologies and showed that findings in the research community have impact on the development of these technologies [16] [17].
- Rombach et al. investigated successes in the practice of software inspections, reviews, and walkthroughs and reported how these have been impacted by software engineering research [18].

Studies under the Impact project looked at the current state-of-the-practice and documented how this state-of-the-practice has been affected/influenced by previously done research work. Different from these studies, in our work we are interested in a complementary approach for evaluating a research work early, based on its potential to address developers need when it matures in the future. Rather than starting with the state-of-practice and looking back, in this work, we start with the state-of-research and look forward to see if these studies can potentially impact how developers do things in the future. Both the retrospective Impact project and our future looking project are important pieces of information to assess the health of software engineering research.

Retrospective Studies in Software Engineering. Lavallée and Robillard performed a restrospective study by systematically reviewing existing studies in the field of software process improvement [19]. They obtained a set of research papers to review from representative venues that publish software process improvement studies, grouped these papers, and reported their findings. Other systematic review studies, for example on fault prediction [20], analyzed papers published in an area over several years to review the current state-of-the-art. These papers neither spanned research topics nor did they use practitioner input on the relevance of papers.

Misirli et al. shared their experience in deploying software analytics solutions, in particular effort estimation and defect prediction solutions, in the industry by interviewing 12 practitioners and obtaining their feedback [21]. Other work has focussed on how practitioners perceived specific software enginerring concepts such as coupling [22], bad smells [23], and productivity [24]. Our study investigates a wider range of topics and involves a much larger number of practitioners.

Ranking Studies. Ranking schemes like the work published in the Journal of Systems and Software [25] [26] ranked individuals and institutions based on their publications in various venues (typically several journals). The ranking did not account for any views from practitioners. Similarly, Ren and Taylor [27] provided a ranking of organizations and individuals using papers published in two journals (TSE and TOSEM) and two conferences (ICSE and FSE) where all papers were weighted similarly. Ren and Taylor summarized the various steps in the ranking process as follows (quoted directly from source): "*1) Choose a field, 2) Select representative publication venues for the field, and, optionally assign a weight to each venue, 3) Set the time range for consideration, 4) Assign a score to each published paper, possibly biased by the venue's weight, 5) Divide the score among multiple authors if the paper has more than one author, 6) Sum the scores for each scholar and each institution, and finally, 7) Rank the scholars and institutions based on sums of their scores*" [27].

Our study is significantly different from these ranking studies, our goal is not to rank papers but to understand the perceived relevance of the research ideas to the broader practitioner community to help advance the state-of-the-art in software engineering research. The ranking of papers discussed above has also been subject to criticism [28].

6. CONCLUSION

In this paper, we proposed to collect practitioner feedback through surveys as a fast and lightweight way to get input on what matters for industry. Such data can help conference organizers to assess and improve the relevance of their meetings, authors to improve their research, and practitioners to discover the latest software engineering research. As a proof-of-concept, we performed such a survey at Microsoft. We invited 3,000 practitioners working in various technical roles to provide feedback on software engineering research. From 512 engineers (response rate 17%), we received in total 17,913 ratings and 173 comments on research ideas. We used this data to assess the health of software engineering research, identify important research topics, and common reservations against research results. Our experiment at Microsoft suggests that practitioners are generally positive to studies done by the software engineering research community - 71% of all ratings were positive - but there is room for improvement.

The next step for this work is to replicate our feedback surveys in other *companies* based in various countries. This is important to get a broader view on software engineering research (not just from one company), as relevance of research ideas could vary by *domain* and *geography*. Replicating the surveys for *open source* developers is important too as they often have a different motivation than practitioners in the commercial software industry.

As part of the replication, we believe that there is an opportunity (and need) to assemble a representative panel of practitioners who are willing to provide feedback to software engineering research. Panels are often used in market and user research for surveys. A requirement for such a panel would be that it includes practitioners from different companies, domains, countries, genders, etc. at representative proportions. Once assembled, the panel could also be used for other research surveys.

Lastly, we are interested in repeating the survey by partnering with conference organizers on an ongoing basis to have industry-accessible, vetted, 1-5 sentences summaries of SE research papers, which can then be rated by a wide range of practitioners to assess the relevance of papers at the conference. Repeating this process regularly will help improve the relevance of software engineering research in the years to come.

7. ACKNOWLEDGMENTS

Thanks to everyone who responded to our survey and to Tom Ball, Christian Bird, Prem Devanbu, Miryung Kim, Emerson Murphy-Hill, Andreas Zeller, and the anonymous ESEC/FSE reviewers for providing feedback on this work.

REFERENCES

[1] L. J. Osterweil, C. Ghezzi, J. Kramer and A. L. Wolf, " Determining the Impact of Software Engineering Research on Practice," *IEEE Computer,* vol. 41, no. 3, pp. 39 - 49, March 2008.

[2] B. Vasilescua, A. Serebrenik, T. Mens, M. G. v. d. Brand and E. Pek, "How healthy are software engineering conferences?," *Science of Computer Programming,* vol. 89, pp. 251-272, September 2014.

[3] L. Briand and A. van der Hoek, "Insights and Lessons Learned from Analyzing ICSE 2014 Survey and Review Data," Luxembourg, 2014.

[4] B. A. Kitchenham and S. L. Pfleeger, "Personal Opinion Surveys," in *Guide to Advanced Empirical Software Engineering* , Springer, 2008, pp. 63-92.

[5] D. Lo, N. Nagappan and T. Zimmermann, "Appendix to The Health of Software Engineering Research," Microsoft Research. Technical Report. MSR-TR-2014-119. Available at: http://research.microsoft.com/apps/pubs/?id=228247, Redmond, WA, 2014.

[6] A. Begel and T. Zimmermann, "Analyze This! 145 Questions for Data Scientists in Software Engineering," in *ICSE*, 2014.

[7] S. Xiao, J. Witschey and E. R. Murphy-Hill, "Social influences on secure development tool adoption: why security tools spread.," in *CSCW'14: Proceeding of the Conference on Computer Supported Cooperative Work*, 2014.

[8] J. Witschey, O. Zielinska, A. Welk, E. Murphy-Hill, C. Mayhorn and T. Zimmermann, "Quantifying Developers' Adoption of Security Tools," in *ESEC/FSE'15: Proceedings of joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, Bergamo, Italy, 2015.

[9] D. Spencer, Card Sorting: Designing Usable Categories, Rosenfeld Media, 2009.

[10] SurveyGizmo.com, *Survey Diagnostics,* http://surveygizmov4.helpgizmo.com/help/article/link/diagnostics.

[11] Microsoft, *Global Diversity and Inclusion,* http://www.microsoft.com/about/diversity/en/us/default.aspx.

[12] B. G. Ryder and M. L. Soffa, "Influences on the design of exception handling ACM SIGSOFT project on the impact of software engineering research on programming language design," *ACM SIGSOFT Software Engineering Notes,* vol. 28, no. 4, July 2003.

[13] B. G. Ryder, M. L. Soffa and M. and Burnett, "The impact of software engineering research on modern progamming languages," *ACM Trans. Softw. Eng. Methodol.,* vol. 14, no. 4, pp. 431-477, October 2005.

[14] J. Estublier, D. Leblang, A. Hoek, R. Conradi, G. Clemm, W. Tichy and D. Wiborg-Weber, "Impact of software engineering research on the practice of software configuration management," *ACM Trans. Softw. Eng. Methodol.,* vol. 14, no. 4, pp. 383-430, Oct 2005.

[15] L. A. Clarke and D. S. Rosenblum, "Historical Perspective on Runtime Assertion Checking in Software Development," *SIGSOFT Software Eng. Notes ,* vol. 31, no. 3, pp. 25-37, May 2006.

[16] W. Emmerich, M. Aoyama and J. Sventek, "The Impact of Research on Middleware Technology," *SIGSOFT Softw. Eng. Notes ,* vol. 32, no. 1, pp. 21-46, Jan 2007.

[17] W. Emmerich, M. Aoyama and J. Sventek, " The Impact of Research on the Development of Middleware Technology," *ACM Trans. Softw. Eng. Methodol.,* vol. 17, no. 4, pp. 1-48, Aug 2008.

[18] D. Rombach, M. Ciolkowski, R. Jeffery, O. Laitenberger, F. McGarry and F. Shull, "Impact of research on practice in the field of inspections, reviews and walkthroughs: learning from successful industrial uses," *ACM SIGSOFT Software Engineering Notes,* vol. 33, no. 6, pp. 26-35, Nov 2008.

[19] M. Lavallée and P. N. Robillard, "The impacts of software process improvement on developers: A systematic review," in *ICSE*, 2012.

[20] T. Hall, S. Beecham, D. Bowes, D. Gray and S. Counsell, "A Systematic Literature Review on Fault Prediction Performance in Software Engineering," *IEEE Trans. Softw. Eng,* vol. 38, no. 6, pp. 1276-1304, 2012.

[21] A. T. Misirli, B. Caglayan, A. Bener and B. Turhan, "A Retrospective Study of Software Analytics Projects: In-Depth Interviews with Practitioners," *IEEE Software,* vol. 30, no. 5, pp. 54-61, Sept-Oct 2013.

[22] G. Bavota, B. Dit, R. Oliveto, M. D. Penta, D. Poshyvanyk and A. D. Lucia, "An empirical study on the developers' perception of software coupling," in *ICSE '13: Proceedings of the 2013 International Conference on Software Engineering*, 2013.

[23] F. Palomba, G. Bavota, M. Di Penta and R. Oliveto, "Do They Really Smell Bad? A Study on Developers' Perception of Bad Code Smells," in *ICSME'14: International Conference on Software Maintenance and Evolution*, 2014.

[24] A. N. Meyer, T. Fritz, G. C. Murphy and T. Zimmermann, "Software developers' perceptions of productivity," in *FSE 2014: Proc of the Intl. Symposium on Foundations of Software Engineering*, 2014.

[25] W. Wong, T. Tse, R. Glass, V. Basili and T. Chen, "An Assessment of Systems and Software Engineering Scholars and Institutions (2002-2006)," *Journal of Systems and Software,* vol. 82, no. 8, pp. 1370-1373, 2009.

[26] W. Wong, T. Tse, R. Glass, V. Basili and T. Chen, "An assessment of systems and software engineering scholars and institutions (2003-2007 and 2004-2008)," *Journal of Systems and Software,* vol. 84, no. 1, pp. 162-168, 2011.

[27] J. Ren and R. N. Taylor, "Automatic and versatile publications ranking for research institutions and scholars," *Commun. ACM ,* vol. 50, no. 6, pp. 81-85, 2007.

[28] D. L. Parnas, "Stop the numbers game," *Commun. ACM ,* vol. 50, no. 11, pp. 19-21, 2007.