# Mastering Variation in Human Studies: The Role of Aggregation

JANET SIEGMUND, Chemnitz University of Technology
NORMAN PEITEK, Leibniz Institute for Neurobiology
SVEN APEL, Saarland University
NORBERT SIEGMUND, Leipzig University

The human factor is prevalent in empirical software engineering research. However, human studies often do not use the full potential of analysis methods by combining analysis of individual tasks and participants with an analysis that aggregates results over tasks and/or participants. This may hide interesting insights of tasks and participants and may lead to false conclusions by overrating or underrating single-task or participant performance. We show that studying multiple levels of aggregation of individual tasks and participants allows researchers to have both, insights from individual variations as well as generalized, reliable conclusions based on aggregated data. Our literature survey revealed that most human studies perform either a fully aggregated analysis or an analysis of individual tasks. To show that there is important, non-trivial variation when including human participants, we reanalyze 12 published empirical studies, thereby changing the conclusions or making them more nuanced. Moreover, we demonstrate the effects of different aggregation levels by answering a novel research question on published sets of fMRI data. We show that, when more data are aggregated, the results become more accurate. This proposed technique can help researchers to find a sweet spot in the tradeoff between cost of a study and reliability of conclusions.

CCS Concepts: • **General and reference** → **Empirical studies**; **Evaluation**; **Experimentation**; • **Human-centered computing** → *Empirical studies in HCI*.

Additional Key Words and Phrases: Human studies, Data aggregation, Guidelines

**ACM Reference Format:**
Janet Siegmund, Norman Peitek, Sven Apel, and Norbert Siegmund. 2018. Mastering Variation in Human Studies: The Role of Aggregation. *ACM Trans. Softw. Eng. Methodol.* 1, 1 (September 2018), 39 pages. https://doi.org/10.1145/1122445.1122456

## 1 INTRODUCTION

Empirical methods have become standard for evaluating new approaches in software-engineering research: Of 1584 papers published in major software-engineering venues between 2011 and 2018, an overwhelming number of 1579 papers conducted some kind of empirical evaluation, ranging from case studies to controlled experiments (see Section 3). Likewise, the human factor in software-engineering research has grown more and more important: Between 1993 and 2002, only 1.9 % of all empirical studies included human participants [68]. In 2010, this number has increased to 18 % [10], and between 2011 and 2018, it increased further to 25 %.

Authors' addresses: Janet Siegmund, janet.siegmund@informatik.tu-chemnitz.de, Chemnitz University of Technology; Norman Peitek, Norman.Peitek@lin-magdeburg.de, Leibniz Institute for Neurobiology; Sven Apel, apel@cs.uni-saarland.de, Saarland University; Norbert Siegmund, norbert.siegmund@informatik.uni-leipzig.de, Leipzig University.


1049-331X/2018/9-ART $15.00
https://doi.org/10.1145/1122445.1122456

Conducting and analyzing human studies is a complex and time-consuming endeavor. The distinguishing element are the human participants with their high variability in task performance, even when all participants originate from a rather homogeneous group (e.g., students of the same semester should have comparable programming experience [23]). The main cause for variation are latent constructs that cannot be measured directly, but only based on indirect indicators (e.g., comprehensibility of source code or programming experience). These constructs lead to variations within and between *human participants* performing *tasks* in an empirical study, affecting the applicability of study results. This has implications for the design and analysis of such studies. As example, suppose we want to evaluate a tool that has been developed to improve bug fixing. We could design a single bug-fixing task to test the research hypothesis that the new tool improves bug fixing productivity compared to an existing one, or we could pose different, but similar bug-fixing tasks to answer this one research hypothesis (similar to the study discussed in Section 4.5). We could analyze the results of each task individually (i.e., task-wise), or summarize the performance of participants over all tasks (e.g., the mean response time for all tasks). Accordingly, we could analyze the performance of each participant (i.e., participant-wise analysis), or compute the mean response time over all participants to answer the research hypothesis. Summarizing the responses times over all tasks and over all participants at the same time would be also possible.

Interestingly, psychology and social sciences have long been using more than one task or indicator to ensure valid and reliable measurement of human participants [55]. Well-known examples include the measurement of intelligence[1] or personality according to the big-five model[2], which is based on a questionnaire of over 50 questions, where several questions are used to describe one of the five factors. To design such tests, a series of many studies with many different participants was necessary, such that the concrete application scenarios in terms of participant groups and measured constructs are well-understood.

An ideal software-engineering study evaluating a new tool or approach would include human participants and involve a *series* of tasks (e.g., 5 bug-fixing tasks) and an analysis that considers both, individual tasks and performances and an *aggregated* view over all tasks and participants to cancel out individual task variation [15]. Aggregation in this context means that several data points are combined to another, representative value by applying an aggregation function [30] (more on this in Section 2.2). This can happen along the dimension of human participants as well as of tasks, or both. By applying both, a *per participant/per task* analysis and an aggregated analysis, researchers can study the effect of *individual participants* and *tasks* on the results, and at the same time also draw general conclusions about the new tool or approach.

In this paper, we take a closer look at aggregation and how it can change the results of software engineering studies. The overall goal of this paper is to raise awareness that the question of whether or not to aggregate observational data in empirical software engineering is not trivial and should be considered already during the design phase of a study, in addition to during the analysis phase. To fulfill our goal, we employ three different methods: First, we conducted a literature survey of the past eight instances of the International Conference on Software Engineering (ICSE), the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), and the Empirical Software Engineering Journal (EMSE) as primary venues for empirical research on software engineering. With this analysis, we obtain an overview of how empirical studies are typically analyzed, that is, per task or aggregated over (categories of) tasks. In summary, 22 (of 144) studies used a task-wise analysis, 71 conducted an aggregated analysis, and 51 used a combined approach. Thus, the question of whether to aggregate or not to aggregate is relevant.

Second, we reanalyzed studies that conducted a task-wise analysis to understand how aggregation can affect the results. To this end, we aggregated the response data of 12 papers and reran the analysis on the data that we aggregated. In a nutshell, we found instances where the aggregation changed the conclusions (6) and where

[1] See,e.g.,http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0103-166X2015000100013
[2] http://psych.colorado.edu/~carey/courses/psyc5112/readings/psnstructure_goldberg.pdf

it led to the same conclusions (6). So, in half of the cases analyzed, aggregation changed at least partially the conclusion, indicating that researchers need to be aware of the effects.

Third, we trained a classifier on the data of three previous fMRI studies in which participants completed different kinds of tasks [58, 62, 64]. The classifier should predict the kind of tasks that participants were completing, similar to the study by Floyd and others [24]. To evaluate how aggregation affects the accuracy of the classifier, we applied an aggregation function to the data and fed both, the aggregated data and raw data, to the classifier. With the aggregated data, the classifier performed better, which contradicts the assumption that more data means higher accuracy. In other words, aggregation also affects results in this case. Furthermore, we defined intermediate aggregation levels, such that a subset of tasks is aggregated. The more tasks are aggregated, the better the prediction accuracy. This helps researchers to select a sweet spot between designing more tasks for an empirical study and accuracy of results.

By combining these three methods, we use a triangulation approach, such that we first show the prevalence of how often researchers face the decision of whether to aggregate or not to aggregate. Subsequently, with the reanalysis of human studies and re-using the data of our fMRI studies, we demonstrate the effect of aggregation on the results and conclusions. In one case, we obtained even a reversed result. Thus, we make the community aware that the decision of whether and how data should be aggregated is not trivial and should be considered carefully.

In summary, we make the following contributions:

We make researchers and experimenters of human studies aware of the influence of task-specific and participant-specific variation on task performance. To this end, we use a triangulation approach:

- We start by showing that, in the literature, researchers who let participants complete tasks (165) favor an aggregated approach (71 times, i.e., 43 %), often use a combined approach of task-wise and aggregated analysis (51, i.e., 31 %), and also use a task-wise analysis only (22, i.e., 13 %; Section 3).
- We provide evidence that data aggregation can affect the conclusions of a human study. To this end, we reanalyze archival data of 12 different papers. In a nutshell, aggregation can change the conclusions of a paper or make the conclusions more nuanced (Section 4).
- We define different levels of aggregation and demonstrate their effect on reliability and accuracy of study results. In summary, the coarser the aggregation level, the better the prediction accuracy, so aggregation does not necessarily lead to loss of information (Section 5).

We share further insights on the soundness of data analysis in empirical software engineering. Most noteworthy, we found that the necessity of adjusting the $p$ level for multiple comparisons does not seem to be standard in the research community yet.

We support replication by providing relevant material at https://github.com/brains-on-code/conducting-and-analyzing-human-studies. The fMRI data can be provided only upon request to ensure anonymity of participants according to the General Data Protection Regulation (GDPR).

To emphasize the possible impact of our findings for empirical research on human participants: (i) The validity of conclusions drawn from existing papers based on individual tasks might be threatened by task-specific and/or participant-specific properties and variation, (ii) replication of such existing studies might convey different findings, and (iii) studies involving human participants can increase their validity and reliability when multiple similar tasks are designed to answer a single research question.

## 2 BACKGROUND AND RELATED WORK

### 2.1 Designing Empirical Studies

Designing empirical studies such that they address the intended research questions or research hypotheses is not a trivial job and requires considerable effort. For example, we designed a study in which we evaluated

the effect of highlighting configuration choices in source code with background colors on maintainability (cf. Section 4). To evaluate this, we needed to define tasks that capture the process of maintenance. To this end, we injected bugs that participants had to locate and suggest a fix for. At the same time, we needed to make sure that participants mostly work with colored source code to evaluate their effect. So, the bugs needed to be located in colored source code, limiting locations to inject bugs, and the task description had to guide the participants to look at the according source code, without giving too much hint to possible solutions. Furthermore, not to occupy participants for too long, the source code could not be too large. Additionally, we needed tasks that are similar to measure maintainability in a valid and reliable way, but at the same time different enough that no learning effects occur. Similar and even more severe constraints likely exist for other studies, which makes the creation of tasks time consuming and tedious. Thus, when designing empirical studies, we always need to make a compromise between reliability and validity of measurement, on the one hand, and the effort of task creation, on the other hand. Aggregation may help to make a decision on how to make this compromise, as we show in this paper.

## 2.2 Aggregation

Although aggregation appears to be intuitive, it is an overloaded term. To avoid misunderstandings, we focus here on the definition provided by Grabisch and others: "Aggregation is the process of combining several numerical values into a single representative value, and an aggregation function performs this operation" [30]. Depending on the underlying scale type, we define concrete aggregation functions. With a metric scale type, the most intuitive aggregation function is the arithmetic mean. For example, many studies collect response time data, and to combine the response times of different tasks, we use the arithmetic mean. With an ordinal scale, or rank data, we use the median as representative value. With a nominal or categorical scale, we count frequencies and use the sum as representative, so that, for example, the sum of all correct and incorrect answers is used.

Both mean and median are measures of central tendency, which are suitable to describe data and compare groups. This is the objective in many studies, so these measures are suitable for our case. However, they are limited for measuring the underlying latent constructs, such as programming experience or maintainability of software. For operationalizing latent constructs, more sophisticated techniques are necessary, such as factor analysis [12], multiple regression [13], or structural equation modeling [19]. For example, to measure programming experience, we have conducted a principal component analysis to extract latent factors based on answers in a questionnaire [23, 63]. Still, since we want to compare the results between different studies, most of which make comparisons between groups, computing the mean, median, and sum are the suitable aggregation functions for this study.

In Table 1, we illustrate how one can aggregate over tasks, over participant, and over both at the same time. We show hypothetical results of a bug-fixing study, in which 4 expert and 4 novice programmers should each complete 3 bug-fixing tasks. Now, one can aggregate over the participants per group (gray cells) and compare the performance between expert and novices for each task (task-wise analysis). In this case, we would lose the potentially interesting information that Expert 3 is actually a pretty fast participant, while Novice 4 seems to be particularly slow. Alternatively, one could aggregate the performance over the tasks (blue cells) and compare the performance per participant (participant-wise analysis). However, this way, we would lose the potentially interesting information that Task 3 seems to be particularly difficult. Last, one could look at the value that represents the aggregation over tasks and participants at the same time (yellow cells), ignoring task-specific and participant-specific information at the same time. However, if we do want to generalize the results to different kinds of tasks and different participants, this might be a good choice, as we can give information about the average case of a performance benefit, average here referring to average participants and average tasks (which of course makes sense only with a larger sample and more tasks). Thus, the decision of whether and how to

Table 1. Illustration of aggregation per participant (i.e,. participant-wise ), per task (i.e., task-wise ), and a combination of both . The values represent fictional response times [min] of four novice and four expert participants to 3 tasks.

| Participant | Task 1 | Task 2 | Task 3 | Total |
|---|---|---|---|---|
| Expert 1 | 5 | 2 | 8 | 5 |
| Expert 2 | 4 | 4 | 4 | 4 |
| Expert 3 | 2 | 3 | 4 | 3 |
| Expert 4 | 5 | 4 | 6 | 5 |
| Experts | 4 | 3.25 | 5.5 | 4.25 |
| Novice 1 | 3 | 6 | 9 | 6 |
| Novice 2 | 3 | 3 | 3 | 3 |
| Novice 3 | 6 | 4 | 5 | 5 |
| Novice 4 | 8 | 6 | 10 | 8 |
| Novices | 5 | 4.75 | 6.75 | 5.5 |

aggregate data is not trivial and should always be considered carefully, since this decision can significantly affect the results, as we illustrate in the next section.

## 2.3 Related Work

Since we make a methodological contribution, we can relate our work to similar methodological studies. Closest to our work is the study by Larsson and others, who collected publicly available data from published studies and evaluated whether and how outliers were handled [47]. They found that authors rarely describe how outliers were managed, threatening the replicability of data analysis. Another related study is conducted by Kosti and others, who compare the explanatory power of archetypes and average analysis in the context of personality types [43]. They found archetypes to be an intuitive way to interpret different personality types. These archetypes are also a way to aggregate data, just like we demonstrated the effect of different aggregation levels.

There is also work on the status of empirical research in software engineering. The work by Tichy and others can be seen as the starting point, in which they found that software engineering and computer science in general are reluctant to conduct empirical evaluations [70]. The team around Glass, Ramesh, and Vessy conducted a more detailed and diverse evaluation of the study, for which they build a classification scheme to evaluate the research of computing disciplines, separated by Information Systems [72], Software Engineering [29], and Computer Science [27]. In addition to looking at how topics are distributed over the three disciplines, they also looked at research methods and approaches. They found that research in information systems was mostly driven by empirical methods, whereas software engineering and computer science were more concerned with technological advancement than evaluation (with or without human participants) [28]. The lack of empirical research was confirmed by Sjøberg and others, who found that only 1.9 % of the papers reported a controlled or quasi-experiment with human participants [68]. However, they did not include all kinds of empirical research with human participants, so the amount of empirical studies could have been actually higher. Buse and others found that, between 2000 and 2010, studies with human participants were on the rise from 3.5 % to 18 %, and further found that according papers have a higher chance of being accepted at top venues [10]. We found an increase to 25 % between 2011 and 2018, with human participants, and most papers (99 %) did some kind of empirical evaluation.

There is a wide range of guidelines on how to conduct and report empirical research in software engineering. Basili and others developed the goal question metric approach to help researchers to define their empirical

work [5]. Kitchenham and others proposed guidelines on how to conduct systematic surveys and systematic literature surveys [40, 41]. Ko and others propose guidelines on how to evaluate tools with human participants [42], and Sjøberg and others suggest to recruit professional software developers more often for empirical studies [67]. Siegmund and Schumann provide a list of confounding factors that need to be controlled for when conducting empirical studies related to program comprehension [22]. Siegmund and others collected the opinion of the software engineering research community on internal and external validity in empirical studies, and found that there is no consensus on how to conduct and evaluate such studies [65]. We add to these guidelines by recommending to carefully consider data aggregation of tasks and participants.

Finally, there are two studies that applied a machine-learning approach on fMRI data in software engineering. First, Floyd and others conducted an fMRI study, in which participants comprehended source code, reviewed source code, or read prose [24]. Their classifier could predict the kind of condition, which is modulated by programmer expertise. Second, Ikutani and others let programmers with different levels of expertise categorize a program according to the class of algorithm (e.g., sorting, search). Their classifier could also predict the category of a program, which is also modulated by expertise. Different to our methodological work showing the increasing accuracy with a coarser aggregation level, both studies showed that it is possible to predict the kind of condition based on fMRI data.

## 3 THE ROLE OF THE HUMAN FACTOR IN EMPIRICAL SOFTWARE ENGINEERING RESEARCH

To demonstrate that researchers who work with human participants often face the decision of whether or not to aggregate data, we conducted a literature survey of the past eight years (i.e., 2011 to 2018) of ICSE, ESEC/FSE, and EMSE as primary venues for empirical research on software engineering. This included 1584 papers. While completeness is well beyond the scope of this article, this literature survey provides sufficient insight into the state of the art of empirical software engineering research.

Specifically, we analyzed each paper regarding whether

(1) it contains an empirical study (1579)
(2) it included human participants (397)
(3) tasks were included (165)
(4) tasks were conducted that could be aggregated (144)
(5) tasks were aggregated (71), not aggregated (22), or both (51)

We (i.e., two authors of the paper) analyzed each of the 1584 papers manually. We started by reading the title and abstract to determine whether it reports on an empirical study. If it was not apparent, we skimmed the paper and searched with keywords that indicate an empirical study, that is: empirical, study, experiment, participant, subject, human, student, novice, professional, and developer. We applied a very wide definition of empirical close to the origin of the word, that is, "knowledge gained from experience" (from the Greek word "empeiría"). Thus, we include case studies on a single software system or small user studies. An example is the paper by Jha and Mahmoud, who invited participants to create a ground truth to test a classifier to evaluate their approach of categorizing and summarizing reviews of apps [37]. As another example, Kula and others used a survey to complement their analysis of how developers updated their library dependencies [46]. Although both do not include full-scale studies, they still collect data to evaluate their approach. Note that, due to the large number of papers that we analyzed manually, we checked only for a few random papers per year and conference whether they match the selection criteria; we found no disagreement. After this step, we found that 1579 of the 1584 papers contained an empirical study. This is in line with previous surveys [65, 66], witnessing the high awareness of the necessity for empirical evaluation. In Table 2, we summarize the results of the literature survey.

Next, we read the description of the study design to determine whether human participants were involved. When there were unclear cases, we (the two authors) discussed them until reaching consensus. In a few cases,

Table 2. Number of papers (Column "Papers") that include an empirical study (column "Empirical") with human participants (column "Human"), in which tasks were defined (column "Tasks") and different analysis approaches were applied, such that the data was aggregated over tasks (column "Aggregated"), analyzed task-wise (column "Single"), or both (column "Both"). In a few cases, a single task was defined, tasks were too different to be aggregated, or defined as independent variable (all summarized in column "Other")

| Venue | Year | Papers | Empirical | Human | Tasks | Aggregated | Single | Both | Other |
|---|---|---|---|---|---|---|---|---|---|
| ICSE | 2011 | 61 | 61 | 13 | 3 | 1 | 0 | 2 | 0 |
| | 2012 | 82 | 82 | 24 | 14 | 8 | 3 | 2 | 1 |
| | 2013 | 106 | 106 | 36 | 9 | 3 | 2 | 3 | 1 |
| | 2014 | 99 | 96 | 28 | 15 | 6 | 0 | 5 | 4 |
| | 2015 | 84 | 83 | 19 | 9 | 5 | 1 | 2 | 1 |
| | 2016 | 101 | 101 | 26 | 8 | 2 | 1 | 4 | 1 |
| | 2017 | 68 | 68 | 20 | 10 | 6 | 3 | 1 | 0 |
| | 2018 | 105 | 105 | 29 | 9 | 2 | 4 | 1 | 2 |
| FSE | 2011 | 34 | 34 | 2 | 0 | 0 | 0 | 0 | 0 |
| | 2012 | 34 | 34 | 5 | 2 | 1 | 0 | 0 | 1 |
| | 2013 | 51 | 51 | 2 | 2 | 0 | 2 | 0 | 0 |
| | 2014 | 61 | 60 | 13 | 3 | 0 | 1 | 2 | 0 |
| | 2015 | 74 | 74 | 18 | 7 | 1 | 3 | 3 | 0 |
| | 2016 | 73 | 73 | 11 | 3 | 1 | 0 | 1 | 1 |
| | 2017 | 72 | 72 | 9 | 4 | 2 | 1 | 1 | 0 |
| | 2018 | 61 | 61 | 10 | 2 | 0 | 0 | 2 | 0 |
| EMSE | 2011 | 25 | 25 | 7 | 2 | 1 | 0 | 0 | 1 |
| | 2012 | 24 | 24 | 5 | 3 | 1 | 0 | 1 | 1 |
| | 2013 | 31 | 31 | 6 | 3 | 2 | 1 | 0 | 0 |
| | 2014 | 53 | 53 | 29 | 16 | 14 | 1 | 1 | 0 |
| | 2015 | 50 | 50 | 20 | 13 | 9 | 0 | 3 | 1 |
| | 2016 | 64 | 64 | 15 | 7 | 1 | 1 | 4 | 1 |
| | 2017 | 78 | 78 | 24 | 9 | 3 | 0 | 6 | 0 |
| | 2018 | 93 | 93 | 26 | 9 | 2 | 0 | 4 | 3 |
| Combined | | 1584 | 1579 | 397 | 165 | 71 | 22 | 51 | 19 |

human participants were recruited to create a ground truth for a classifier [37]. In other cases, the data of human participants were collected via a tool or plugin [2]. We labeled these studies still as human studies, because data of humans were used to answer the research questions. With this definition, 397 studies relied on human data. This shows that the importance of the human factor in software engineering research has reached the research community. While between 1993 and 2002, only 1.9 % of the papers reported an experiment with human participants [68][3], this number increased to 25 % between 2011 and 2018.

Furthermore, we determined whether tasks were part of the study. From this step on, two authors checked the categorization of each other and discussed unclear cases. In 165 studies, participants completed a kind of task,

[3]However, this study focused only on controlled experiments and did not include any empirical study, as we did. Thus, the actual number of papers including human participants may have been higher.

such as program comprehension [63], refactoring [25], or specification reviews [39]. Researchers design tasks around every-day activities of software developers to evaluate the effect of tools or approaches on these activities.

Next, we analyzed whether tasks can be reasonably aggregated: For example, some tasks are too different to be aggregated, as in the study of Hu and others [33], in which participants should first, create a requirement specification document, and, second, inspect it. In these cases, we cannot be sure whether they measure the same construct or different constructs, so an aggregation might not be suitable. We make this decision based on the task description, not the variance in the response of the tasks. In other cases, there was just a single task or one task per condition, which typically lasted the entire study duration. For example, Endrikat and others evaluated how participants use API documentation, and created one task in which participants use APIs and which lasted several hours [21]. Overall, we found the tasks of 144 studies could be reasonably aggregated, so in 36 % of human studies, researchers face the decision of data aggregation.

Last, to understand the analysis and how the conclusions were drawn, we read the analysis, discussion, and conclusion of each paper. We found that, in 71 studies, task performance was aggregated (e.g., to the number of fixed bugs over all tasks [1] or in terms of average response time across tasks [53]). We also found 22 studies that did not aggregate data, but discussed the results only task-wise, indicating that a tool or approach made a difference for some tasks, but not for other tasks. Additionally, 51 studies used a combined approach of aggregation and task-wise analysis.

To summarize, when it comes to measuring human performance, researchers often face the question of whether to aggregate data. In the next section, we demonstrate the effect of aggregation by reanalyzing 12 papers.

## 4 THE EFFECT OF AGGREGATION ON RESULTS: REANALYSIS OF FOUR PUBLISHED STUDIES

Having shown that the issue of data aggregation is not a corner case, we demonstrate that the choice of aggregation can change the conclusions drawn from studies, stressing the importance of this decision for reaching valid and reliable conclusions. To this end, we revisited each paper that reported on a task-wise analysis. Of all these 22 papers, 9 provided their raw data either directly in the paper or on a supplementary Web site that was still active. For the remaining papers, we contacted the authors: 5 responded (within a day), and 8 did not respond at all (see Section 4.7 for more details). In the end, we could reanalyze 12 papers.

### 4.1 Overview

For each study, we started by extracting the data either from the Web site or directly from the paper and store them in a CSV file. Then, we applied the aggregation function to aggregate similar tasks. Thus, we did not aggregate the data of all tasks, but adhered to the categories that were described in the paper. For example, the study by Barik and others described 5 categories to which their 10 tasks belong [4], so we aggregated the data per category and looked at the 5 categories. Then, we reran the analysis of the authors, for which we used Python or R (all scripts and aggregated data are available on the Web site). When the analysis of the authors did not follow standard statistical procedure, we also ran a further analysis that adhered to the standard. Specifically, for metric data, we first checked for a normal distribution, and in case it was violated, used a Wilcoxon test, and a t-test otherwise. Furthermore, when multiple comparisons were made, we always adjusted the $p$ level for multiple comparisons with a *false-discovery-rate (FDR)* correction [7] to avoid an inflation of the probability of the type-I error (i.e., rejecting the null hypothesis although it is true). For example, with ten tests, the probability increases from 5% to 40%, which can be corrected by the FDR procedure. Essentially, the FDR correction divides the $p$ level by the number of positives tests, which typically leads to a lower $p$ value to be able to reject the null hypothesis. We also applied an FDR correction to the original task-wise analysis of the authors, in case it was not included.

With the reanalysis of the aggregated data, we revisited the research hypotheses and answers. We found that either the conclusions changed after the reanalysis (6 papers), or that the conclusions stayed the same (6 papers).

However, there are some nuances, for example, that the conclusions change because of a specific task, or because of inconsistencies in the original analysis.

We select two papers for each category and describe the studies and reanalysis in detail. All remaining papers are summarized in the appendix.

### 4.2 Same Results: Drag-and-Drop Refactoring: Intuitive and Efficient Program Transformation

Lee and others describe an approach and tool (DNDRefactoring) for drag-and-drop refactoring, and compare their tool to the standard refactoring workflow in Eclipse. The authors found differences in response times between refactoring with Eclipse and DNDRefactoring for all but one refactoring task [49].

Independent variable: Refactoring approach (2 levels, operationalized with Eclipse and DNDRefactoring)
Tasks: 7 refactoring tasks in 4 categories:
- Extract Method (1 task)
- Move Method (4 task)
- Three-step refactoring (i.e., anonymous class to nested + move type to new file + move class) (1 task)
- Extract class (1 task)

Dependent variable: Task completion time [metric scale]
Null hypothesis:
- There is no difference in response times between Eclipse and DNDRefactoring

Results:
- For the extract method refactoring, there is no difference. For all other refactorings, there is a significant difference in favor of DNDRefactoring

In our reanalysis, we evaluated for each task category whether there was a significant effect. Specifically, we conducted a t-test for the three-step task, and a Wilcoxon test for all other tasks, since the according data deviate from a normal distribution. Furthermore, we adjusted the significance level with an FDR correction. In Table 3, we summarize the data and results of the significance tests. With our task-wise analysis, we come to the same conclusion as the authors, namely that, except for the extract method refactoring, DNDRefactoring reduces the response times of the participants to complete the refactoring tasks. We found one issue with the analysis, namely that the authors did not correct for multiple testing, but after applying FDR correction, the results did not change. In summary, we found that aggregating the data did not change the results.

### 4.3 Same Results: Do Developers Read Compiler Error Messages?

In the study by Barik and others, developers' gaze behavior was observed with eye tracking [4]. The authors evaluated how different categories of errors affect the gaze behavior of participants. We shortly summarize the results[4]:

Independent variable: Error categories (5 levels, derived from frequent errors of a large set of builds)
Tasks: 10 tasks to identify a defect based on a compiler error message
Dependent variables:
(1) Correctness, i.e., whether a provided solution was correct or incorrect, or whether there was a timeout (i.e., participants did not provide a solution within a time limit) [nominal scale]
(2) Gaze behavior, i.e., amount of time participants' gaze was on a certain area of the IDE used for the study. Areas are the source code editor, error areas (error popup, problems pane, quickfix popup), and navigation areas (explorer pane, outline pane) [metric scale]

Research questions:

[4]Link to data: http://static.barik.net/barik/gazerbeams/

Table 3. Task completion times for DNDRefactoring. We ran a t-test for the three-step task (Task 7) and total results, and a Wilcoxon test for all other tasks. Gray cells contain values computed by us.

| Participant | Refactoring | Task 1: Extract Method | Task 2: Move Method | Task 3: Move Method | Task 4: Move Method | Task 6: Move Method | Task 7: Three-step Refactoring | Task 8: Extract Class | Total |
|---|---|---|---|---|---|---|---|---|---|
| #1 | Eclipse | 42.3 | 48.3 | 22.9 | 18.5 | 16.6 | 29.9 | 42.1 | 219.8 |
| | DNDR | 18.6 | 12.7 | 13.4 | 13.8 | 14.3 | 20.9 | m iss | 93.7 |
| #2 | Eclipse | 106.4 | 71 | 10.7 | 7.3 | 4.4 | 113.4 | 32.5 | 345.7 |
| | DNDR | 13.8 | 6.7 | 2.8 | 13.2 | 4.9 | 22.8 | 17.6 | 81.8 |
| #3 | Eclipse | 18.6 | 65.4 | 10.7 | 8.1 | 4 | 23.9 | 151.9 | 282.6[1] |
| | DNDR | 33.5 | 3.7 | 2.9 | 2.3 | 1.4 | 16.8 | 8.9 | 69.5 |
| #4 | Eclipse | 53.3 | 11.7 | 18.4 | 13.1 | 4 | 87.1 | 40.5 | 228.1 |
| | DNDR | 55.9 | 5.8 | 3.5 | 2.1 | 2.3 | 39.5 | 23.5 | 132.6 |
| #5 | Eclipse | 23.7 | 93 | 8.9 | 29.8 | 8.2 | 95.9 | 41.5 | 301 |
| | DNDR | 63.1 | 5 | 6.4 | 1.8 | 2 | 13.9 | 11.4 | 103.6 |
| #6 | Eclipse | 10 | 100.5 | 3 | 10.2 | 2.7 | 100.4 | 24.2 | 251 |
| | DNDR | 31.3 | 26.8 | 1.5 | 1 | 1.1 | 15 | 15.3 | 92 |
| #7 | Eclipse | 22.6 | 46.3 | 2.8 | 10.7 | 5.2 | 69 | 25.1 | 181.7 |
| | DNDR | 22.8 | 3.4 | 1.6 | 1.6 | 0.9 | 23 | 7.6 | 60.9 |
| #8 | Eclipse | 18.8 | 136.7 | 4.1 | 6.7 | 2.7 | 77.5 | 23.7 | 270.2 |
| | DNDR | 17.6 | 1 | 2.1 | 1.7 | 4.1 | 6.0 | 21.8 | 55.1 |
| #9 | Eclipse | 7 | 50.7 | 3.1 | 4.7 | 2.3 | 43 | 24 | 134.8 |
| | DNDR | 12.6 | 1.5 | 1.5 | 1.6 | 1.0 | 13.7 | 12.9 | 45.4 |
| Average | Eclipse | 33.6 | 69.3 | 9.4 | 12.1 | 5.6 | 71 | 45.06[1] | 246.1 |
| | DNDR | 29.9 | 7.4 | 4 | 4.3 | 3.6 | 19.2 | 14.9 | 81.6 |
| Speed up[3] | | 1.1 | 9.4 | 2.4 | 2.8 | 1.5 | 3.7 | 4.5 | 3.0 |
| $p$ value (aggr.) | | 0.715 | | **0.002** | | | **0.002** | **0.004** | **0.004** |
| Speed up (aggr.) | | 1.1 | | 5 | | | 3.7 | 4.5 | 3.0 |
| W/t value | | 18 | 45 | 45 | 40 | 41 | 4.81 (df=8) | 36 | 8.17 (df=8) |
| $p$ value | | 0.652[2] | **0.004** | **0.004** | **0.039** | **0.027** | **0.001**[2] | **0.008**[2] | **0.000004**[2] |

[1]These values are reported as 228.1 (total) and 66.75 (average) in the original study, which is incorrect according to the raw data of the study (Table IV in the original paper).
[2]The $p$ values differ from the original study, but we cannot be sure why. Part of the explanation is that we used a t test instead of a Wilcoxon test for the three-step refactoring and the total response time.
[3]Eclipse average divided by DNDRefactoring average, as in the original study.

*RQ corr.*: How effective and efficient are developers at resolving error messages for different categories of errors?
*RQ ∂aze*: Do developers read error messages?
Results: (mostly verbal description)

Table 4. Number of correctly identified defects and gaze area of what participants looked at for each task as well as aggregated for all tasks. We ran a $\chi^2$ test for correctness and an ANOVA for gaze behavior. All statistical values in gray cells are computed by us.

| | | **Correctness** | | | **Gaze Behavior** | | |
|---|---|---|---|---|---|---|---|
| Category | Task(s) | Correct | Incorrect | Timeout | Source | Error | Navigation |
| **Semantic** | Task 1 | 2 | 47 | 6 | 66 | 23 | 10 |
| | Task 2 | 1 | 49 | 4 | 66 | 23 | 11 |
| | Task 9 | 28 | 2 | 25 | 80 | 13 | 7 |
| **Dependency** | Task 3 | 30 | 0 | 25 | 74 | 15 | 11 |
| | Task 4 | 36 | 10 | 9 | 79 | 14 | 6 |
| **Type mismatch** | Task 5 | 49 | 5 | 1 | 68 | 23 | 9 |
| | Task 6 | 55 | 0 | 0 | 65 | 25 | 11 |
| **Syntax** | Task 10 | 50 | 5 | 0 | 65 | 20 | 15 |
| **Other** | Task 7 | 22 | 23 | 10 | 78 | 15 | 7 |
| | Task 8 | 48 | 5 | 2 | 68 | 17 | 15 |
| Semantic | 1, 2, 9 | 31 | 98 | 35 | 70.67 | 19.67 | 9.33 |
| Dependency | 3, 4 | 66 | 10 | 34 | 76.5 | 14.5 | 8.5 |
| Type Mismatch | 5, 6 | 104 | 25 | 1 | 66.5 | 24 | 10 |
| Syntax | 10 | 50 | 2 | 25 | 65 | 20 | 15 |
| Other | 7, 8 | 70 | 28 | 12 | 73 | 16 | 11 |
| | $\chi^2$ / F | | 198.16 | | | 4.624 | |
| | $p$ value | | 0.000 | | | 1 | |

*RQ corr.*: Solution for correctness is skewed; significantly different solution times for correct and incorrect answers

*RQ ∂aze*: Participants' gaze is most of the time on the source-code area (65 % to 80 %), then the error area (13 % to 25 %)

In Table 4, we show the results of individual tasks and summarize them for a reanalysis across all error categories. As aggregation function, we applied the sum for correctness and the mean for the gaze behavior. We aggregated all tasks of each category according to Table 2 of the original paper. Then, we were able to conduct a more nuanced analysis. For correctness, the category of error affected the number of correct solutions (*RQ correctness*). This was especially apparent for the semantic category (many incorrect responses) and the type-mismatch category (many correct responses). With a task-wise analysis, this influence of the different categories of error messages did not become clear, but it revealed only that different tasks affected correctness. With aggregation, we found that semantic error messages may be problematic, but error messages regarding type errors do not seem to pose much of a challenge. Furthermore, we can state that the category of error does not affect the gaze behavior (*RQ ∂aze*), such that participants spent a large amount of time on the source code. In contrast to the original study, we can narrow the amount of time from 65 % to 80 % down to 65 % to 76.5 %, and from 13 % to 25 % down to 16 % to 24 %, giving a more specific estimation of participants' gaze behavior. Thus, with the aggregation, a more nuanced analysis of the effect of different kinds of errors on programmer behavior was possible.

### 4.4 Differing Results: Do Background Colors Improve Program Comprehension in the #ifdef Hell?

We have conducted a study to evaluate the effect of background colors on maintenance in configurable software [22]. We shortly summarize the important aspects of the study[5]:

Independent variable: Variability annotations for configurable software (2 levels, operationalized with background colors and `#ifdefs`)
Tasks: 6 program comprehension tasks in two categories (plus a warm-up task that was not analyzed)
– Feature location, referred to as static (2 tasks)
– Bug location, referred to as maintenance (4 tasks)
Dependent variables:
(1) Task completion time [metric scale]
(2) Correctness of a provided solution (correct, incorrect) [nominal scale]
Null hypotheses:
$H_0 static$: The kind of annotation does not affect task completion time for static tasks
$H_0 maintenance$: The kind of annotation does not affect task completion time for maintenance tasks
$H_0 correctness$: The kind of annotation does not affect correctness
Results:
$H_0 static$: Significant difference for the static tasks regarding task completion time
$H_0 maintenance$: Significant difference for one of the maintenance tasks regarding task completion time
$H_0 correctness$: No significant difference regarding correctness

In Table 5, we summarize the results of the reanalysis across all task categories. As aggregation function, we applied the mean for the response times and the sum for correctness. We aggregated the 2 static tasks and the 4 maintenance tasks, as both constitute different categories of task with our analysis, and come to the same conclusion that background colors lead to shorter task completion times for static tasks ($H_0 static$), and that the kind of annotation does not affect the correctness for any of the tasks ($H_0 correctness$). However, we reject $H_0 maintenance$, indicating an effect of the kind of annotation on the completion time for maintenance tasks. Looking at the direction of the effect, participants with background colors are significantly slower than participants with `#ifdefs`. However, this conclusion would be incorrect, because one particular maintenance task was substantially different than the others, so it skewed the results. In this specific task, participants had to work with a class that was entirely annotated with a red background color, causing visual fatigue and slowing down participants. Thus, with a blind aggregation, we would arrive at the incorrect conclusion that background colors slow down participants for maintenance tasks. We pick up on the discussion of choosing a suitable level of aggregation in Section 6.

### 4.5 Differing Results: Clone-Based and Interactive Recommendation for Modifying Pasted Code

Lin and others developed an approach (CCDemon) to automatically recommend whether and where pasted code should be edited [50]. In a study, they compared their approach to a state-of-the-art baseline, MCIDiff.

Independent variable: Tool support for working with code clones (2 levels, opertionalized with MCIDiff and CCDemon)
Tasks: 6 programming tasks in 3 levels of complexity (low, medium, high)
Dependent variables:
(1) Task completion time [metric scale]
(2) Number of failed tests [nominal scale]
Null hypotheses:

[5]Link to data: https://www.infosun.fim.uni-passau.de/se/janet/colors/index.php

Table 5. Task completion time and correctness for static and maintenance tasks. We ran a $t$ test for task completion time and $\chi^2$ test for correctness. All statistical values in gray cells are computed by us (average/sum row are computed based on the raw data).

| **Kind of Annotation** | **Static Tasks** | | | **Maintenance Tasks** | | |
|---|---|---|---|---|---|---|
| | Task | Time [s] | Correct / Incorrect | Task | Time [s] | Correct / Incorrect |
| Background Colors | S1 | 426 | 12 / 10 | M1 | 414 | 21 / 1 |
| | S2 | 282 | 14 / 8 | M2 | 342 | 21 / 1 |
| | | | | M3 | 468 | 19 / 3 |
| | | | | M4 | 1404 | 15 / 7 |
| Average/sum | | 352.86 | 26 / 18 | | 657.13 | 76 / 12 |
| `#ifdefs` | S1 | 738 | 7 / 14 | M1 | 432 | 19 / 2 |
| | S2 | 372 | 12 / 9 | M2 | 354 | 19 / 2 |
| | | | | M3 | 396 | 12 / 9 |
| | | | | M4 | 882 | 12 / 9 |
| Average/sum | | 554.12 | 19 / 23 | | 515.99 | 62 / 22 |
| t test/$\chi^2$ | | 3.93 | 1.14 | | 2.49 | 3.52 |
| $p$ value | | 0.000 | 0.285 | | 0.012 | 0.061 |

(1) Tool support does not affect task completion time
(2) Tool support does not affect the number of failed tests

Results:

(1) Significant difference in completion time for three of the tasks
(2) No significant difference in failed tests for any of the task

In our reanalysis, we aggregated the tasks by complexity as indicated in Table 6 of the paper, so Tasks 1 and 2 are summarized to low complexity, Tasks 3 and 4 to medium complexity, and Tasks 5 and 6 to high complexity. In Table 6, we summarize the results of our reanalysis.

For low and high complexity, we find a significant difference in terms of response time in favor of CCDemon. Regarding failed test cases, we could confirm the results of the authors that there is no difference between the two approaches. Furthermore, we found inconsistencies in the analysis of the authors. First, the authors applied a Wilcoxon test, which is not the optimal choice neither for response times nor for failed test cases. For response times, they are interval scaled and for low and high complexity, normally distributed, and variance homogeneity also holds. Hence, at least for low and high complexity, a t test would have been the better choice, because it has more statistical power. For completeness, we have added a t test to Table 6, and the significant difference for the high-complexity task barely holds. Regarding test cases, since these are frequency data, a $\chi^2$ test would have been the more fitting choice. We also added this to the table, and we observe a significant difference for high complexity tasks, but in favor of the control group (i.e., the control group produces fewer test case failures). Thus, we would conclude that the approach of the authors leads to more errors.

Another reason for our deviating results is that the authors did not correct for multiple testing [3]. Applying an FDR correction to the task-wise analysis of the authors would make all significant differences vanish [7]. For our reanalysis on the aggregated data, however, one difference would still remain. With the FDR correction, the difference for the low complexity task still holds, showing that the approach of the authors makes participants significantly faster for low complexity tasks. Thus, in this case, aggregating the data for this task would make a stronger statement in favor of the approach of the authors.

Table 6. Reanalysis of the study on tool support for code clones. We ran a $t$ test for completion time, $\chi^2$ test for test case failures, and as an alternative a two-way ANOVA with participant group and task complexity as factors. Column "Wilcoxon Test" contains the results of the Wilcoxon test as reported by the authors. All values in gray cells are computed by us

| **Approach** | | **Completion Time** | | | **Wilcoxon Test** | | **Test Case Failures** | | | **Wilcoxon Test** | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Low | Medium | High | Z | p | Low Mean/Sum | Medium Mean/Sum | High Mean/Sum | Z | p |
| CCDemon | Task 1 | 64.25 | | | 2.52 | 0.01 | 0 / 0 | | | 0.00 | 1 |
| | Task 2 | 91.63 | | | 2.10 | 0.04 | 0 / 0 | | | 1.00 | 0.32 |
| | Task 3 | | 390.13 | | 0.14 | 0.89 | | 1.13 / 9 | | 0.09 | 0.93 |
| | Task 4 | | 287.75 | | 0.42 | 0.67 | | 0.25 / 2 | | 0.38 | 0.71 |
| | Task 5 | | | 141.88 | 2.34 | 0.02 | | | 0.38 / 3 | 1.00 | 0.32 |
| | Task 6 | | | 346.88 | 0.70 | 0.48 | | | 2.25 / 18 | 1.12 | 0.26 |
| | Aggregated | 77.94 | 338.94 | 244.38 | | | 0 / 0 | 0.69 / 11 | 1.31 / 21 | | |
| MCIDiff | Task 1 | 147.50 | | | | | 0 / 0 | | | | |
| | Task 2 | 181.25 | | | | | 0.13 / 1 | | | | |
| | Task 3 | | 463.13 | | | | | 2.13 / 17 | | | |
| | Task 4 | | 260.00 | | | | | 0.63 / 5 | | | |
| | Task 5 | | | 278.75 | | | | | 0 / 0 | | |
| | Task 6 | | | 405.00 | | | | | 1.00 / 8 | | |
| | Aggregated | 164.38 | 361.56 | 341.88 | | | 0.06 / 1 | 1.38 / 22 | 0.5 / 8 | | |
| | W | 5.5 | 71 | 23 | | | 0 | 30 | 23 | | |
| | $p$ value | 0.001 | 0.90 | 0.038 | | | 1 | 0.820 | 0.143 | | |
| | t/$\chi^2$ | 4.473 | | 2.115 | | | 1 | 3.667 | 5.828 | | |
| | df | 15 | | 15 | | | 1 | 1 | 1 | | |
| | $p$ value | <0.001 | | 0.051 | | | > 0.05 | >0.05 | <0.05 | | |
| | ANOVA | Effect of Group: F: 3.663 p: 0.059<br>Effect of Complexity: F: 14.648 p < 0.001<br>Complexity * Group: F: 0.421 p: 0.658 | | | | | | | | | |

Going on step further, one could even conduct a two-way ANOVA with complexity as second factor. In this case, the approach would have no significant effect, and instead, the effect of task complexity would be the only significant effect. So, our reanalysis shows how important it is to choose the correct analysis method. We can either show that the tool of the authors has a positive or no effect (results of the Wilcoxon tests and t tests), that the approach has a negative effect ($\chi^2$ test of test case failures), or that the complexity of the tasks seems to be the determining factor in the response time differences (ANOVA). The best conclusion that we can draw here is that the data are inconclusive and that we would need further studies to provide a more definitive answer.

## 4.6 Summary

Our reanalysis and discussion of these 4 studies, together with reanalysis the remaining 8 studies summarized in the appendix, illustrates that aggregation over multiple tasks can lead to contrary results compared to a task-wise analysis. On the one hand, we showed that an aggregation of tasks can remove task-specific variation, which can

Table 7. Summary of the four re-analyzed studies presented in this paper. A summary of all re-analyzed studies can be found on our project Web site.

| Sec. | Statistical Test? | Correction for Multiple Comparison? | Appropriate Statistical Tests? | Test for Normality? | Different results? |
|---|---|---|---|---|---|
| Drag-and-Drop Refactoring: Intuitive and Efficient Program Transformation | | | | | |
| 4.2 | Yes | No. Results would stay the same | Yes | Unclear ("we cannot assume that the data (configuration time) is normally distributed") | No |
| Do Developers Read Compiler Error Messages? | | | | | |
| 4.3 | Partly for response times, but not for correctness or gaze behavior | N/A | N/A | N/A | No, but more nuanced |
| Do Background Colors Improve Program Comprehension in the `#ifdef` Hell? | | | | | |
| 4.4 | Yes | No. Results would change for one task | Yes | Yes | Yes (task-specific peculiarity) |
| Clone-Based and Interactive Recommendation for Modifying Pasted Code | | | | | |
| 4.5 | Yes | No. Results would change for two tasks | Partly | No | Yes. Data can be interpreted in different directions. |

bring more robust results. On the other hand, aggregation may be sensitive to largely deviating data of single participants or tasks, especially for the arithmetic mean. Thus, studies with human participants may benefit from a combined approach of aggregated and task-wise analysis. Specifically, it might make sense to define up front which tasks are similar and can be aggregated. Then, the data of each task can be analyzed to observe whether the tasks are indeed similar. If the data indicate that one task is different, then this task should not be included in the aggregation.

### 4.7 Further Insights

In addition to the results regarding how aggregation affects conclusions of a research paper, we obtained some further insights that we would like to share. First, from 22 papers that reported a task-wise analysis, only 9 papers provided publicly available data. For the others, either no link was provided, the link was broken, or did not contain the raw data, but rather high-level results, the description of tasks, or a developed tool. As a side note, the missing data shed some light on the difficulty of proper replication in software engineering [52]. An interesting story is by the authors of the paper "Feature Model Extraction from Large Collections of Informal Product Descriptions" [16]. The feature models were generated from code as an extremely large visual tree which ran the whole way down the hallway when printed out. By the time we requested the data, it was too late to create a shareable version (e.g., as XML model), because the servers that stored the data had been impounded by the FBI

to catch a crime ring, and so they are not available.[6] This unlikely event illustrates that anything can happen to the data, so researchers need a safe and reliable way to store data.

Furthermore, we found that often, we cannot replicate the exact $p$ values of the authors. We suspect the reason for this to be different versions of statistical software and packages. While often, this might not be a problem, it might lead to different conclusions when $p$ values are close to the defined significance level. In this case, it might make sense to also rely on effect size and statistical power before accepting or rejecting a hypothesis.

Another issue that we found in some papers is that statistical procedures are not applied entirely correctly. This confirms the findings by Reyes and others, who found that the software engineering research community seems to lack standard statistical knowledge [60]. The most frequent inconsistency that we found (12 papers) was that the significance level was not adjusted for multiple comparisons. The problem is that the probability of a type-I error increases with the number of tests [3]. Thus, the significance level needs to be reduced depending on the number of tests, for example, with an FDR correction[7]. In few cases, we also found that not the optimal statistical test was applied (e.g., a Wilcoxon test on frequency data). On few occasions, a more powerful ANOVA could have been applied, which would lead to more informed insights on which factors affect the study results. Thus, although there has been much progress when it comes to applying statistical analysis procedures, there is still room for improvement in the software engineering research community.

## 5 LEVELS OF AGGREGATION: TOWARD A COMPROMISE BETWEEN RELIABILITY AND EFFORT

In this section, we take a different perspective on how aggregation can affect the results of a study with human participants. Instead of rerunning analyses on aggregated data, we use a reverse approach in that we use the data to train a classifier. To this end, we are using the data of three fMRI studies and apply different levels of aggregation to the fMRI data, train a classifier for each level, and compare the performance of all trained classifiers. The different aggregation levels demonstrate that we can combine the benefits of aggregation with a task-wise/participant-wise analysis to find a sweet spot between effort of task creation and reliability of measurement. We start with an introduction to fMRI to understand why such data are especially suitable to demonstrate the effect of aggregation.

### 5.1 Background on fMRI

In a nutshell, fMRI measures the brain activation over time. To this end, it exploits the different magnetic properties of oxygenated and deoxygenated blood. When a brain region activates, its oxygen need increases, so the amount of oxygenated blood increases and the amount of deoxygenated blood decreases, referred to as the BOLD (blood oxygenation level dependency) response [35]. To measure it, an fMRI scanner takes a snapshot (referred to as *scan*) of the brain typically every two seconds. The spatial resolution of this scan is defined in terms of voxels, which are cubes with $1mm$ or $3mm$ edge length. With $3mm$ voxels, there are over 100 000 voxels for the entire brain.

When completing a task, not the entire brain is recruited, but only a subset, depending on the task. For example, in the three fMRI studies of which we use the data [24, 58, 62, 64], five brain regions were significantly activated to comprehend source-code snippets as compared to finding syntax errors in the same code snippet, while other regions remain unchanged or become deactivated, meaning that the amount of oxygenated blood decreases. This deactivation typically happens in the default mode network, which is active during self-referential processing [54]. For this self-referential processing not interfering with a cognitive task, the default mode network deactivates during a task.

[6] Thanks so much to the authors for sharing this anecdote with the community!

[7] Another option is the Bonferroni correction, but its focus is too much on avoiding the type-I error [59]

For classification, we used only those voxels that show a significant signal change, because only these contribute to a task and are relevant to observe the effect of aggregation. Specifically, we look at the set of activated voxels, the set of deactivated voxels, and the union of both sets. This way, we reduce the noise that irrelevant voxels may cause, and we reduce the input for the learning pipeline from more than 100 000 voxels to 400 to 2 000 voxels.

Having discussed the technical background of fMRI, we now dive into our research objective to evaluate the effect of different levels of aggregation, introducing further information on fMRI as needed.

## 5.2 Defining Aggregation Levels

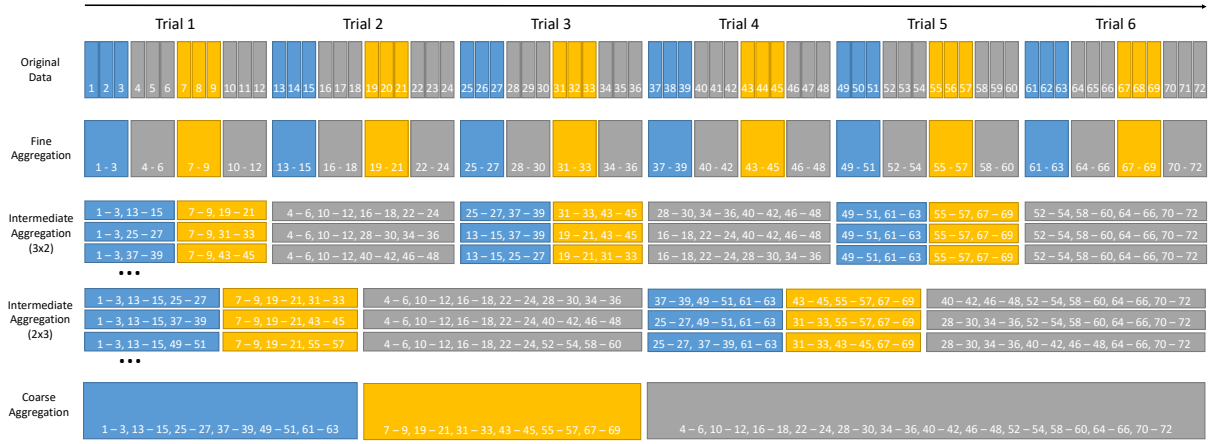



Fig. 1. Visualization of fine, intermediate, and coarse aggregation levels. Blue boxes represent brain scans of comprehension tasks, yellow boxes represent brain scans of syntax tasks, and gray boxes represent brain scans of rest.

To obtain a deeper understanding of the effect of aggregation on experimental results, we introduce *levels* of aggregation. We start with a *coarse* aggregation level, that is, based on an entire series of tasks. To this end, we show an example fMRI study in Figure 1, which contains a time series of *one* $3mm$ voxel. In the example, there are 3 different conditions (comprehension (blue), rest (gray), syntax (yellow)), and 3 scans for each condition (a scan is a snapshot of the brain typically taken every two seconds). The conditions in fMRI studies are summarized to *trials*, each containing a different, but similar source code snippet. One trial is a sequence of comprehension, rest, syntax, and rest (intermittent rest conditions are typical for fMRI studies to allow the brain activation to return to baseline). Thus, scans 1 to 12 comprise trial 1. For the coarse aggregation, we compute the arithmetic mean of the signal of all scans (having a metric scale) that belonged to one condition (e.g., scans 1 to 3, 13 to 15, 25 to 27, 37 to 39, 49 to 51, and 61 to 63 for comprehension). Since there are 3 different conditions, we have 3 different data points, that is, the average value of the condition comprehension, syntax, and rest, respectively (bottom row in Figure 1).

Next, we define the *fine* aggregation level, in which we aggregate only those scans of one task that are taken consecutively, so we summarize the first 3 scans to comprehension, the next 3 (4 to 6) to rest, the next 3 (7 to 9) to syntax, and the next 3 (10 to 12) to rest. We repeat this for each trial, resulting in 6 trials 4 conditions = 24 data points (second row in Figure 1).

Now, typical fMRI studies consist of more than 6 trials. For example, the study presented at ICSE 2014 consisted of 12 trials with 4 conditions each (i.e., comprehension, rest, syntax, rest) [62]. For each of the 12 trials, a different source code snippet was used. Thus, with the fine aggregation, there are 48 data points per participant (i.e.,

12 trials  4 conditions = 48), while for the coarse aggregation, there are still 3 data points per participant (because there are 3 different conditions, overall). Between 3 data points and 48 data points, there are lots of other options of aggregation, so we define further levels of aggregation.

Coming back to the example of Figure 1, we could also aggregate the value of 2 trials to groups (compared to all trials (coarse) and for each trial (fine)). For instance, we can aggregate trials 1 and 2, trials 3 and 4, and trials 5 and 6. Alternatively, we can aggregate trials 1 and 3, and 2 and 4, and 5 and 6, or trials 1 and 4, 2 and 3, and 5 and 6. In Figure 1 (third row), we illustrate these three options, but there are 30 different options. Another aggregation level is to aggregate the values of 3 trials to groups. We can aggregate trials 1, 2, and 3 to one group and trials 4, 5, and 6 to a another group; or, trials 1, 2, 4, and 3, 5 and 6, or trials 1, 2, 5, and 3, 4, 6. Again, we depict these 3 options in Figure 1 (fourth row), but there are 10 different options in total. With the actual fMRI studies, we can define even more intermediate aggregation levels. For example, for the study presented at ICSE, we aggregated to 2, 3, 4, and 6 groups. For each intermediate level, we randomly created 10 variants of how trials are assigned to groups.

## 5.3 Objective

In this section, we want to demonstrate the effect of different aggregation levels on the quality of results. With fMRI having found its way into software-engineering research, we have the opportunity to use data that was collected in a highly controlled and repetitive setting. This brings several advantages. First, participants received straight forward and basic instructions, for example, to comprehend a source code snippet, to locate a syntax error in a source code snippet, or to rest, all for a short interval of 30 to 60 seconds. Thus, we can be fairly certain at any point in time what participants were doing, giving us a ground truth. Second, to reliably capture fMRI data, the same conditions were repeated several times, however, with different, yet similar source code snippets. It is important to note that these source code snippets were designed to be homogeneous, that is, we intended to exclude the influence of peculiarities of individual snippets. The repetition of conditions provides us with a large data base of several hundred scans per participant, giving us several thousand data points for each study to demonstrate the effect of different aggregation levels. In this sense, our results are limited to similar studies that also produce large data sets, which are needed for building classifiers.

## 5.4 Operationalization

To demonstrate the effect of different aggregation levels, we apply a classification approach to predict the condition (i.e., rest, comprehension, syntax). The classifier receives the fMRI data sets as input with varying degrees of aggregation levels, that is, from coarse over intermediate levels to a fine level, finishing with no aggregation at all (see Section 5.5 for details). The rationale of using a classifier is to have an unbiased predictor, taking every information—useful and misleading—into account that is provided during learning. By controlling the training set in terms of aggregation levels, we obtain a clear picture about the effect of different aggregation levels on prediction accuracy, allowing us to demonstrate to the effect of data aggregation on study results. The higher prediction accuracy, the more suitable is the underlying level of aggregation as input for the classifier.

## 5.5 Material

As material, we used the data of three fMRI studies [58, 62, 64] summarized in Table 8. In each study, we observed that several brain areas showed a significant signal change, either in terms of an increase (i.e., they activate) or decrease (i.e., they deactivate). As input for the classifier, we consider all voxels that show a significant signal change. Specifically, we use the activated voxels, the deactivated voxels, and their union as input for the classifier, after applying the different levels of aggregation. We did not include all voxels (more than 100 000 for the entire brain), because the voxels without a significant signal change do not show sufficient variation to be used as input

Table 8. Overview of the 3 fMRI studies.

| Study | ICSE '14 [62] | FSE '17 [64] | ESEM '18 [58] |
|---|---|---|---|
| Participants | 16 | 14 | 17 |
| Trials | 12 | 30 | 25 |
| Conditions | Comprehension, Rest, Syntax | Comprehension, Rest, Syntax | Comprehension, Attention, Rest, Syntax |
| Scans | 900 | 900 | 838 |
| Voxels activation | 446 | 390 | 1606 |
| Voxels deactivation | 966 | 1798 | 3459 |
| Voxels union | 1412 | 2188 | 5065 |
| Data points for fine aggregation | 48 | 54 | 60 |
| Tasks per intermediate aggregation levels | 6, 4, 3, 2 | 13, 9, 3, 2 | 10, 5, 4, 2 |
| Data points for intermediate aggregation levels | 8, 12, 16, 24 | 4, 6, 18, 26 | 6, 12, 15, 30 |
| Size of training set/test set | 12 / 4 | 11 / 3 | 13 / 4 |

for the classifier. Based on the voxels, the classifier should predict the condition, that is, comprehension, rest, syntax, or attention. In Table 8, we summarize relevant data for all 3 studies.

We have described the ICSE 2014 study when introducing the aggregation levels (Section 5.2), so we continue here with the second study (i.e., FSE 2017). 11 participants took part, but 3 participants were measured twice with a different set of source-code snippets. Since they saw different snippets, we decided to treat them as different participants, so we assume to have 14 participants in this sample. A further change compared to the ICSE study was the design of the tasks. This study contained 30 trials of 2 tasks each, and one task was either comprehension or syntax-error finding, and the other task was rest. There were 27 trials with comprehension, and only 3 trials with syntax-error finding, which we excluded because they do not provide sufficient data. The 27 comprehension trials varied in terms of semantic information of the snippets, but since these differences were not reflected in the activated areas (only in the strength of activation), we do not make this distinction in the analysis. Thus, we have 27 trials of 2 tasks each (i.e., comprehension and rest). We defined four aggregation levels between the coarse and the fine aggregation: We aggregated to 3 and to 9 groups, and to 2 and 13 groups, where we randomly excluded one trial for each of the 10 variants that we created for each intermediate aggregation level.

The third study (i.e., ESEM 2018) was a non-exact replication of the FSE study with 17 participants. It used the same source-code snippets, but contained some changes to the experiment design. Specifically, an attention task[8] was added after each comprehension task to reduce comprehension-related activation during rest. Overall, there were 20 trials of three tasks each: comprehension, attention, and rest. Again, for the learning step, we excluded five trials of syntax error finding due to the small amount of data. We aggregated the trials to 2, 4, 5, and 10 groups.

All available data sets have been preprocessed with BrainVoyager™ QX 2.8.4[9] following the standard pipeline: 3D-motion correction, slice-scan-time correction, and temporal filtering. Moreover, to account for anatomical differences between participants' actual brains, we transformed the anatomical scan of each participant into the

[8]The d2 task, in which participants should respond to all d's with two markings in a sequence of similar visual stimuli [9].
[9]Brain Innovation B.V., Netherlands, https://www.brainvoyager.com

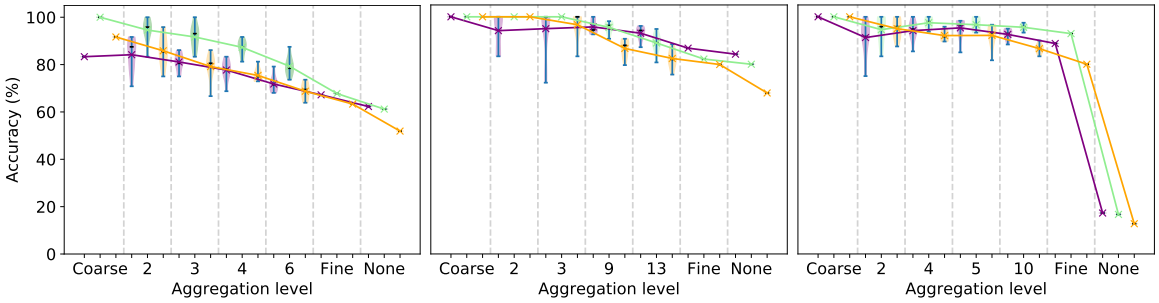



Fig. 2. Accuracies of the classifier for the ICSE (left), FSE (center), and ESEM study (right). Each figure shows different aggregation variations, from a coarse aggregation on the left to no aggregation on the right. The numbers denote to how many groups data were aggregated. The colors denote the set of voxels; purple: activated voxels, yellow: deactivated voxels, green: union.

standard Talairach brain [69]. Finally, the functional data of each participant were spatially smoothed with a Gaussian filter (FWHM=4 mm).

We normalized the data of each participant with a z transformation. That is, from each signal for each voxel, we subtract the mean for each participant and divide it by the standard deviation of each participant. Furthermore, we replaced outliers by the mean of the 2 consecutive voxels. A value is an outlier when it deviates more than certain multiples of the standard deviation from the mean, which translated to 4 multiples for the ICSE set, and 6 for both, the FSE and ESEM sets (a plot of the normalized values of the voxels before and after outlier removal is available at the project's Web site). This procedure of outlier removal also highlights the cost of aggregating data across studies, as we need to take a close look at the data and cannot use the same value for the data of all studies.

## 5.6 Methods

For training the classifier, we used the preprocessed data sets as described above. We created a different data set for each aggregation level as input for the classifier. This translated to 7 input data sets for each study: coarse aggregation, 4 levels of intermediate aggregation, fine aggregation, and no aggregation. The classifier predicted the kind of task that participants were completing, based on the signals of the voxels. Thus, the features for classification are the voxels, and the labels are the conditions. To find the optimal machine-learning pipeline including optimal hyper parameters, we used TPOT (https://github.com/EpistasisLab/tpot) [48], which applies evolutionary algorithms to find the best classifier and hyper-parameters via cross validation [56]. Thus, for the actual learning step, we used different approaches, such as Random Forest Classifier [8] and Support Vector Machines [14]. The rationale of optimizing the classifier for each aggregation level is that we do not want to bias certain levels of aggregation due to an improper selection of classifier or hyper-parameters, eliminating a threat to internal validity (cf. Section 7). The replication package contains the specific approach and hyper parameters for each data set.

There are several ways to split the data into training and test set: Randomly, by snippet, or by participant (which we selected). First, a random split would rule out systematic errors when done a sufficient number of times, but it would pose a problem for the non-aggregated data. It could well be that, for consecutive data points, one is in the training set and the following in the test set. However, since consecutive data points are more similar than non-consecutive data points (since the BOLD response takes a few seconds to manifest and return to the baseline), this would overestimate the accuracy of the classifier. In other words, the classifier would have seen all

tasks, all participants, and all snippets during learning and would not be confronted with anything new during prediction. Thus, we eliminated this threat to construct validity.

Next, selecting different snippets for the training and test set would be possible, since the snippets were designed to be similar and thus should elicit a comparable brain activation. However, this would not be possible for the coarse aggregation, because we aggregate the data over all snippets.

To have the same type of split for all aggregation levels, we split the data by participants. To this end, we use the data of 75 % of participants for the training set and 25 % of the data for the test set.

## 5.7 Results

Next, we present the results of our analysis. In Figure 2, we show the accuracies of all 7 aggregation levels. Each of the 3 plots is ordered according to the aggregation levels, starting left from the coarse level, over the intermediate levels of aggregation, to the fine aggregation and no aggregation on the right. For the coarse, fine, and no aggregation levels, we have only one accuracy result, because there is no variation in the aggregation levels (e.g., for the coarse aggregation, we summarize all trials of one task per participant). Since we randomly generated several variants for the intermediate aggregation levels, we have different data points for each aggregation level resulting in different classification accuracies, which we illustrate with violin plots.

For the ICSE study, the worst prediction accuracy is for the non-aggregated data, which is just above 50 %, so in about half the test data, an incorrect condition is predicted. Since we have 3 different conditions, the baseline (i.e., guessing) would mean 33 % accuracy. Hence, the worst accuracy is better than guessing. For the fine aggregation, the prediction accuracy increases to 63 %, and for the coarse aggregation, we have the highest prediction accuracy of 83 % and even up to 100 %. Looking at all the aggregation levels, the accuracies almost follow a linear behavior, such that with a decreasing level of aggregation, the prediction accuracies decrease, too. The only exception is for the activated areas, where the accuracies are slightly better when aggregating to 2 groups than to 1 group (i.e., the coarse level). Nevertheless, the general picture of decreasing accuracy with decreasing level of aggregation is apparent.

For the FSE study, we observe a similar trend, but the prediction accuracies should be higher, because we predict only 2 conditions (i.e., comprehension and rest), so 50 % accuracy would be guessing. The worst accuracy predictions are at about 68 % for the deactivated areas without aggregation. As for the ICSE study, the general picture of higher prediction accuracy with higher aggregation levels still holds, such that with the coarse aggregation level, we have the best prediction accuracy of 100 %. With decreasing aggregation levels, the accuracy decreases. There actually seems to be a clear drop between the aggregation levels, but it occurs between 3 groups and 9 groups. Thus, with an additional aggregation level between these two, this drop might not be this apparent.

For ESEM, the worst prediction results also occur without aggregation, with only 13 % for the deactivated areas. This is far below the guessing baseline of 33 %, and we suspect one reason to be the similar activation pattern for both, the attention task and the rest task. As soon as we aggregate the data, the accuracy considerably increases to 80 %, and also reaches 100 % for the coarse aggregation level. The ESEM data are similar to the ICSE and FSE data, except for the steep drop below the guessing baseline without any aggregation. Here, there does not seem to be a linear relationship, but rather a sweet spot, such that, as soon as the data are aggregated, the prediction level increases and stays at a high level.

## 5.8 Discussion

The classifier that we trained on the fMRI data showed that, over all data sets, the higher the aggregation level, the higher the accuracy of the learned model. In other words, the coarser level of aggregation led to better results, showing again that aggregation can make a considerable difference for the results. Furthermore, we found evidence for a linear relationship (ICSE, FSE) and a sweet spot of aggregation (ESEM), which is a good message

to determine a priori (e.g., in a pre-study) the cost–benefit tradeoff of an experiment having a series of tasks: Up until a certain point, more tasks increase reliability, but beyond that, the benefit of higher validity may not be worth the additional cost of creating tasks. Thus, we could easily limit the number of tasks to a manageable number and still have certain confidence about the reliability and validity of the drawn conclusions.

An important insight is that, although there is more information available when no aggregation is performed (i.e., we have more data points on which the classifier can learn on), there is less accuracy in the predictions. Considering the central limit theorem, which predicts that an aggregation of means of many samples is a good approximation of the true mean of the population, it seems obvious that the classifier performs better on the aggregated data. By contrast, Simpson's paradox describes that the mean can also be an unreliable predictor, given that different samples have considerably different sizes. Looking closer, the central limit theorem and Tschebyscheff's version of the Weak Law of Large Numbers are statistical effects based on data aggregation over independent runs with identical distributions. However, these two laws of probability theory do not apply here directly: (i) the runs in the fMRI experiment are not all independent, as the activation in a voxel at time t+1 depends also on the activation of the same voxel at time t, and (ii) the number of experiments (and aggregations) are far too low to be affected by the law of large numbers, as the training set size is a function over the number of labels (i.e., the more labels we have, the more data points we need for learning). We would need approximately 70k independent training data points for our labels to be affected by the law of the large numbers, which is far more than the partially dependent 14k data points. So, the prerequisites of the law do not hold.

Moreover, we see here an overlap with statistical learning theory, which quantifies the amount of data needed to accurately and reliably learn a function (i.e., the classifier). Vapnik and Chervonenkis found in 1971 that the training set size depends on the complexity of the function to be learned (i.e., the function parameters) [71]. One needs more data to learn a more complex function. In the case of class imbalance (as we have it), the training data size must scale even more [38]. Since we use only the aggregates for learning, we would expect a lower accuracy for smaller training set sizes.

Hence, we have two statistical observations pointing in opposite directions. Throughout the paper we argue that, despite fundamental results from statistical learning theory [71] and despite the prerequisites of the central limit theorem and the related law of large numbers are not met, authors need to consider aggregation as an additional tool, which might lead to counter-intuitive results, as the Simpson's paradox illustrates [73]. The concrete cause for this observation is, however, unclear and under current research. Theoretical explanations about the effectiveness of bootstrap aggregation (or bagging) [17], which might apply here, are under debate. Some theories point to the direction of noise or variance reduction [6] and others state that the main cause is equalizing leverage points (related to outliers) [31].

## 6 IMPLICATIONS AND GUIDELINES

In the previous sections, we have shown that the role of the human factor for empirical software engineering research is on the rise, and that the decision for aggregation affects the result.

We have uncovered several reasons that are in favor of and against aggregation, which reflect the difficulty of finding a sweet spot to match resource constraints (e.g., in terms of time to prepare a study) and to design an experiment that can produce valid and reliable results.

In a nutshell, the reasons against aggregation include:

> Researchers may fear loss of critical information due to aggregation, because several data points are summarized to few, which could lead to incomplete conclusions.
> Aggregating measures of human performance is not trivial, because there are various options to define an aggregation function, each of which can have a different effect on generalizability and reliability of

results. Selecting an unsuitable aggregation function may lead to the opposite effect of decreasing (instead of increasing) reliability of the drawn conclusions.

Posing multiple tasks for a single research question causes a substantial effort for study planning, execution, and (statistically sound) analysis, as does increasing the number of participants taking part in a study.

Tasks might be too different to be aggregated. However, a wider range of tasks can be more representative for software engineering tasks and create more generalizable results.

Points in favor of aggregation include:

With aggregation, we summarize more indicators for a latent construct, increasing the reliability of its measurement, because task-specific or participant-specific variation are less pronounced (as shown by our classifiers). This way, we borrow from well-established techniques from well-established sciences.

We can make more nuanced statements of how underlying latent constructs can affect results thereby increasing our understanding of how a new tool or approach affects software engineers (as shown by the reanalysis presented in Section 4.3).

Especially when having small samples, aggregation can increase statistical power and increase trust in our conclusions (as shown by the reanalysis presented in Section 4.5).

Thus, both options provide their own advantages and drawbacks.

In our literature survey, we found 22 papers that analyzed the results in a task-wise manner only, although an aggregation would also have been possible. This way, the peculiarities of single tasks cannot be canceled out and might even be pronounced, especially when conclusions are drawn based on one task, while the results of others may be ignored. At the same time, we found that 71 studies applied an aggregation, which also has the potential of biasing the conclusions, as the reanalyzed study on background colors showed: We rejected a null hypothesis, which, however, is actually caused by the peculiarity of one task. Thus, it is important to apply both, an analysis of aggregated data and also of individual tasks and participant data. This is especially important when it seems that individual tasks or participants affect a result considerably. With our study on different levels of aggregation, we raise the awareness that there are various options when it comes to aggregating data of single tasks (or participants), and that a intermediate aggregation might provide a good compromise between accounting for the variation of individual participants and tasks and canceling out the peculiarities of individual participants and tasks to be able to draw general conclusions.

Thus, our results lead to different recommendation regarding aggregation.

Use aggregation to check assumptions about tasks

First, when having designed multiple tasks to measure one construct, such as maintainability, we can evaluate whether our assumption is actually true. For example, in the study on background colors (cf. Section 4.4), we designed the 4 maintenance tasks to be similar. Only after the experiment we did learn that one task was different from the three others. In the literature survey, we found 4 papers where the tasks were too different to be grouped, such that a reanalysis based on an aggregation was not appropriate (e.g., one study included a task to find a bug and another task to describe a possible solution). Thus, the consideration of aggregation can help researchers in finding false assumptions of study designs (e.g., 2 tasks only appear to be similar) and can therefore help experimenters improving the design. Furthermore, a clear reporting helps other researchers replicate empirical studies and be aware of peculiar tasks.

Use aggregation to increase trust in results

Second, aggregation of several tasks can improve reliability and validity of measurement. For example, the study about gaze behavior of developers (cf. Section 4.3) showed a more nuanced picture after tasks of the same category were aggregated. This is common procedure in psychology and social science, and should also be embraced more by software engineering researchers. Especially with a small sample size, an aggregated analysis can increase trust in the results.

Use intermediate aggregation levels to find a compromise
Third, we devised a definition of intermediate aggregation levels, which helps to find a compromise between study effort (in terms of task creation and duration) and reliability and validity of measurement. Typically, tasks in software engineering studies are more time consuming than in psychology (in which aggregation is a standard procedure). For example, it takes a few seconds to answer a question in a big-five personality questionnaire, or a few minutes to complete one task in an intelligence test, whereas bug-fixing tasks or code-inspection tasks last considerably longer. Thus, having 20 tasks in psychology is often not a problem, but it often is a problem in software engineering research (e.g., 3 minutes times 20 would result in a 60-minute study, but, say 5 minutes for a bug-fixing task times 20 tasks would already result in a 100-minute study). To attain a good compromise between reliability and validity of measurement, on the one hand, and effort for designing empirical studies, on the other hand, intermediate aggregation levels can help.

It is important to note that our results must not be seen as invitation to $p$ hacking or finding an aggregation level of data that fits with the expectations. As the reanalysis of the study with the clone detection tool showed, the same data can show that the new tool improves the detection of code clones, makes it worse than a standard tool, or that task difficulty is the main driver for differences in performance, not the tool (cf. Section 4.5). Thus, aggregation levels should be defined upfront: However, when it becomes apparent that the tasks are not as similar as expected, it is acceptable to change aggregation levels and document this change. In this line, it helps to preregister a study (https://osf.io/prereg/), so that the expectations are communicated upfront.

Furthermore, it is crucial to select the appropriate statistical test for an experimental design and the collected data, because otherwise, the data can lead to opposing conclusions. Additionally, the significance level needs to be adjusted when multiple comparisons are made, because this can lead to finding spurious effects. This issue actually led to a doubt of the validity of many fMRI studies [20], and software-engineering researcher should not face the same doubts. A good source for selecting appropriate tests may be a standard statistics books [3] or decision charts for statistical tests[10].

Our analysis also highlights that, for replication studies and meta-analyses, the aggregation function of a primary study plays a crucial role. Especially meta-nalyses are defined by applying an aggregation function to the data published studies, so that results can be summarized and generalized [26]. Our reanalysis of both the studies of the literature survey and the fMRI studies showed that we should not necessarily expect that replications and meta analyses come to the same conclusions as the original studies.

## 7 THREATS TO VALIDITY

### 7.1 Internal Validity

In literature surveys, there is the threat of overlooking relevant papers or selecting inappropriate categories for a paper. To reduce this threat, we manually analyzed each paper and discussed cases with an ambiguous assignment.

As input for learning, we used preprocessed data. We could also use the raw data with a different learning approach, such as convolutional neuronal networks, which compensate for spatial and anatomical differences of participants, as they learn that slightly different positions in two images might still be attributed to the same label, if the pattern is similar [44]. However, we are not aware of their performance for fMRI data, so we selected established methods. Moreover, a similar setting has been used by others [24, 36].

To avoid bias due to unsuitable learning pipelines for fMRI data, we used TPOT to find the optimal pipelines and according hyper-parameters. Thus, we are confident that our results of decreasing accuracy with decreasing level of aggregation are not biased by inappropriate learning pipelines.

[10]Many are available online, e.g., https://stats.idre.ucla.edu/other/mult-pkg/whatstat/, accessed 01/22/20

### 7.2 Construct Validity

To minimize the threat to construct validity caused by systematic bias in training and test set, we avoided a random split, as consecutive data points might be in training and test set. Furthermore, we used the same kind of split (by participants), for all aggregation levels to avoid any differences caused by different kinds of training and test set.

### 7.3 External Validity

To demonstrate the effect of different aggregation levels on prediction accuracy, we used data of three fMRI studies, as they are especially suitable to demonstrate our point. Clearly, this selection limits the generalizability of our results to similar studies. However, due to the controlled nature of the study (keeping everything homogeneous), we expect that, in other studies, in which participants complete similar tasks, such as comprehending source code or fixing a bug, individual variations are even more pronounced. Thus, we are confident that our results are applicable to a wide range of software engineering studies with human participants.

As there are more fMRI studies on tasks related to programming, we could have extended our data set, specifically by studies of Floyd and others [24], Huang and others [34], Krueger and others [45], Duraes and others [18], and Castelhano and others [11]. However, this was not possible, as the data either were collected with a different experimental paradigm or could not be shared due to the General Data Protection Regulation.

## 8 CONCLUSION

With empirical methods having become standard, there are numerous empirical studies with human participants in software engineering research. In this article, we point to a methodological gap when conducting and analyzing human studies, as empirical studies often do not account for individual variations of tasks and human participants, which threatens the validity of the conclusions. The problem is that, on the one hand, those individual variations can reveal interesting insights of tasks and even a misconception of the empirical study design, but, on the other hand, may also lead to false conclusions by overrating a single task or participant performance.

Thus, we proposed to use different *aggregation levels*, a novel variation of traditional aggregation to conduct and analyze empirical studies with human participants. Since the different aggregation levels rely on the principle of statistical sampling to deal with variation, they can increase result quality compared to singleton analysis. Aggregation levels support researchers conducting empirical studies with human participants in the following way: First, in a pre-study, aggregation levels help determining the sweet spot between a suitable number of tasks and participants and reliability of conclusions based on a research question. Second, in the analysis, they offer the best of both worlds, that is, (i) taking into account individual peculiarities of participants and tasks that would have been canceled out by a too coarse aggregation and (ii) providing robust statistical evaluation by means of statistical sampling to manage inter-task or inter-participant variation.

To account for this variation, we propose to aggregate data of individual participants and tasks at different levels to combine the benefits of individual insights and statistical sound conclusions that cancel out individual variations. The results of our literature survey showed that only 51 papers use both, an aggregated and an individual analysis. By reanalyzing 12 studies of our literature survey, we could show that the decision whether to aggregate or not to aggregate data is not trivial and can affect the soundness of conclusions. To further demonstrate the effect of different aggregation levels, we reanalyzed the data of three published fMRI studies on program comprehension. We could increase accuracy of the results with an increasing number of tasks that were aggregated, showing that a coarse aggregation can have considerable benefits. Thus, in this case, the question of whether to aggregate can be answered with yes. However, increasing the number of tasks makes an empirical study costlier and time-consuming. With using aggregation levels in a pre-study, experimenters can find a sweet spot between experimental effort and reliability and validity of the drawn conclusions. In summary, there is a

wide range of possible aggregation levels, and with our work, we aim at making researchers and experimenters aware of the effect of aggregation, and encouraging the research community to use the best of both worlds, that is, combining a per-participant/per-task analysis with an aggregated analysis.

## ACKNOWLEDGMENT

We like to thank André Brechmann for his valuable and thorough feedback on this manuscript. Also many thanks to Westley Weimer for his feedback on an early version. J. Siegmund's work is supported by DFG grants SI 2045/2-2. J. Siegmund's work is further funded by the Bavarian State Ministry of Education, Science and the Arts in the framework of the Centre Digitisation.Bavaria (ZD.B). N. Siegmund's work is supported by DFG grants SI 2171/2 and SI 2171/3. Apel's work has been supported by the German Research Foundation (AP 206/6).

## REFERENCES

[1] Wasif Afzal, Ahmad Nauman, GhaziJuha Itkonen, Richard Torkar, Anneliese Andrews, and Khurram Bhatti. 2015. An Experiment on the Effectiveness and Efficiency of Exploratory Testing. *Empirical Softw. Eng.* 20, 3 (2015), 844–878.

[2] Emil Alégroth, Robert Feldt, and Lisa Ryrholm. 2015. Visual GUI Testing in Practice: Challenges, Problems and Limitations. *Empirical Softw. Eng.* 20, 3 (2015), 694–744.

[3] Theodore Anderson and Jeremy Finn. 1996. *The New Statistical Analysis of Data.* Springer.

[4] Titus Barik, Justin Smith, Kevin Lubick, Elisabeth Holmes, Jing Feng, Emerson Murphy-Hill, and Chris Parnin. 2017. Do Developers Read Compiler Error Messages?. In *Proc. Int'l Conf. Software Engineering (ICSE)*. IEEE CS, 575–585.

[5] Victor Basili. 1992. *Software Modeling and Measurement: The Goal/Question/Metric Paradigm.* Technical Report CS-TR-2956 (UMIACS-TR-92-96). University of Maryland at College Park.

[6] Eric Bauer and Ron Kohavi. 1999. An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants. *Machine Learning* 36, 1-2 (1999), 105–139.

[7] Yoav Benjamini and Yosef Hochberg. 1995. Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing. *Journal of the Royal Statistical Society. Series B (Methodological)* 57, 1 (1995), 289–300.

[8] Leo Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (2001), 5–32.

[9] Rolf Brickenkamp, Lothar Schmidt-Atzert, and Detlev Liepmann. 2010. *Test d2-Revision: Aufmerksamkeits- und Konzentrationstest.* Hogrefe Göttingen.

[10] Raymond Buse, Caitlin Sadowski, and Westley Weimer. 2011. Benefits and Barriers of User Evaluation in Software Engineering Research. In *Proc. Int'l Conf. Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)*. ACM, 643–656.

[11] Joao Castelhano, Isabel Duarte, Carlos Ferreira, Joao Duraes, Henrique Madeira, and Miguel Castelo-Branco. 2018. The Role of the Insula in Intuitive Expert Bug Detection in Computer Code: An fMRI Study. *Brain Imaging and Behavior* 13, 3 (2018), 623–637.

[12] Raymond Cattell. 1952. *Factor Analysis.* New York: Harper.

[13] Jacob Cohen and Patricia Cohen. 1983. *Applied Multiple Regression: Correlation Analysis for the Behavioral Sciences* (second ed.). Addison Wesley.

[14] Corinna Cortes and Vladimir Vapnik. 1995. Support-Vector Networks. *Machine Learning* 20, 3 (1995), 273–297.

[15] Kurt Danziger. 1990. *Constructing the Subject.* Cambridge University Press.

[16] Jean-Marc Davril, Edouard Delfosse, Negar Hariri, Mathieu Acher, Jane Cleland-Huang, and Patrick Heymans. 2013. Feature Model Extraction from Large Collections of Informal Product Descriptions. In *Proc. ACM Joint Symposium Foundations of Software Engineering (FSE)*. ACM, 290–300.

[17] Pedro Domingos. 1997. Why Does Bagging Work? A Bayesian Account and Its Implications. In *Proc. Third International Conference on Knowledge Discovery and Data Mining (KDDâĂŹ97)*. AAAI Press, 155–158.

[18] J. Duraes, H. Madeira, J. Castelhano, C. Duarte, and M. C. Branco. 2016. WAP: Understanding the Brain at Software Debugging. In *Proc. Int'l Symposium Software Reliability Engineering (ISSRE)*. IEEE CS, 87–92.

[19] Rick Hoyle (Editor). 2012. *Handbook of Structural Equation Modeling.* The Guilford Press.

[20] Anders Eklund, Thomas E. Nichols, and Hans Knutsson. 2016. Cluster Failure: Why fMRI Inferences for Spatial Extent Have Inflated False-Positive Rates. *Proc. National Academy of Sciences* 113, 28 (2016), 7900–7905.

[21] Stefan Endrikat, Stefan Hanenberg, Romain Robbes, and Andreas Stefik. 2014. How Do API Documentation and Static Typing Affect API Usability?. In *Proc. Int'l Conf. Software Engineering (ICSE)*. ACM, 632–642.

[22] Janet Feigenspan, Christian Kästner, Sven Apel, Jörg Liebig, Michael Schulze, Raimund Dachselt, Maria Papendieck, Thomas Leich, and Gunter Saake. 2013. Do Background Colors Improve Program Comprehension in the #ifdef Hell? *Empirical Softw. Eng.* 18, 4 (2013), 699–745.

[23] Janet Feigenspan, Christian Kästner, Jörg Liebig, Sven Apel, and Stefan Hanenberg. 2012. Measuring Programming Experience. In *Proc. Int'l Conf. Program Comprehension (ICPC)*. IEEE CS, 73–82.
[24] Benjamin Floyd, Tyler Santander, and Westley Weimer. 2017. Decoding the Representation of Code in the Brain: An fMRI Study of Code Review and Expertise. In *Proc. Int'l Conf. Software Engineering (ICSE)*. IEEE CS, 175–186.
[25] Xi Ge, Quinton L. DuBose, and Emerson Murphy-Hill. 2012. Reconciling Manual and Automatic Refactoring. In *Proc. Int'l Conf. Software Engineering (ICSE)*. IEEE CS, 211–221.
[26] Gene V. Glass. 1976. Primary, Secondary, and Meta-Analysis of Research. *Educational Researcher* 5, 10 (1976), 3–8.
[27] Robert Glass. 1992. A Comparative Analysis of the Topic Areas of Computer Science, Software Engineering, and Information Systems. *The Journal of Systems and Software* 19, 3 (1992), 277–289.
[28] Robert Glass, Venkataraman Ramesh, and Iris Vessey. 2004. An Analysis of Research in Computing Disciplines. *Commun. ACM* 47, 6 (2004), 89–94.
[29] Robert Glass, Iris Vessey, and Venkataraman Ramesh. 2002. Research in Software Engineering: An Analysis of the Literature. *J. Information and Software Technology* 44, 8 (2002), 491–506.
[30] M. Grabisch, J.L. Marichal, R. Mesiar, and E. Pap. 2009. *Aggregation Functions*. Cambridge University Press.
[31] Yves Grandvalet. 2004. Bagging Equalizes Influence. *Machine Learning* 55, 3 (01 Jun 2004), 251–270.
[32] Stefan Hanenberg, Sebastian Kleinschmager, Romain Robbes, Éric Tanter, and Andreas Stefik. 2014. An Empirical Study on the Impact of Static Typing on Software Maintainability. *Empirical Softw. Eng.* 19, 5 (2014), 1335–1382.
[33] Wenhua Hu, Jeffrey C. Carver, Vaibhav Anu, Gursimran S. Walia, and Gary L. Bradshaw. 2018. Using Human Error Information for Error Prevention. *Empirical Softw. Eng.* 23, 6 (2018), 3768–3800.
[34] Yu Huang, Xinyu Liu, Ryan Krueger, Tyler Santander, Xiaosu Hu, Kevin Leach, Tyler Santander, and Westley Weimer. 2019. Distilling Neural Representations of Data Structure Manipulation using fMRI and fNIRS. In *Proc. Int'l Conf. Software Engineering (ICSE)*. IEEE CS, 396–407.
[35] Scott Huettel, Allen Song, and Gregory McCarthy. 2008. *Functional Magnetic Resonance Imaging*. Sinauer Associates.
[36] Yoshiharu Ikutani, Takatomi Kubo, Satoshi Nishida, Hideaki Hata, Kenichi Matsumoto, Kazushi Ikeda, and Shinji Nishimoto. 2020. Expert Programmers have Fine-Tuned Cortical Representations of Source Code. *bioRxiv* (2020). https://www.biorxiv.org/content/early/2020/01/29/2020.01.28.923953
[37] Nishant Jha and Anas Mahmoud. 2018. Using Frame Semantics for Classifying and Summarizing Application Store Reviews. *Empirical Softw. Eng.* 23, 6 (2018), 3734–3767.
[38] Brendan Juba and Hai Le. 2019. Precision-Recall Versus Accuracy and the Role of Large Data Sets. In *Proc. AAAI Conf. on Artificial Intelligence*, Vol. 33. 4039–4048.
[39] Jakub Jurkiewicz, Jerzy Nawrocki, Mirosław Ochodek, and Tomasz Głowacki. 2015. HAZOP-based Identification of Events in Use Cases. *Empirical Softw. Eng.* 20, 1 (2015), 82–109.
[40] Barbara Kitchenham, Pearl Brereton, David Budgen, Mark Turner, John Bailey, and Stephen Linkman. 2009. Systematic Literature Reviews in Software Engineering: A Systematic Literature Review. *J. Information and Software Technology* 51, 1 (2009), 7–15.
[41] Barbara Kitchenham, Pearl Brereton, Zhi Li, David Budgen, and Andrew Burn. 2011. Repeatability of Systematic Literature Reviews. In *Proc. Int'l Conf. Evaluation and Assessment in Software Engineering (EASE)*. IET Software, 46–55.
[42] Andrew Ko, Thomas Latoza, and Margaret Burnett. 2015. A Practical Guide to Controlled Experiments of Software Engineering Tools with Human Participants. *Empirical Softw. Eng.* 20, 1 (2015), 110–141.
[43] Makrina Viola Kosti, Robert Feldt, and Lefteris Angelis. 2016. Archetypal Personalities of Software Engineers and Their Work Preferences: A New Perspective for Empirical Studies. *Empirical Softw. Eng.* 21, 4 (2016), 1509–1532.
[44] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 1097–1105.
[45] Ryan Krueger, Yu Huang, Xinyu Liu, Tyler Santander, Westley Weimer, and Kevin Leach. 2020. Neurological Divide: An fMRI Study of Prose and Code Writing. In *Proc. Int'l Conf. Software Engineering (ICSE)*. 12. To appear.
[46] Raula Gaikovina Kula, Daniel M. German, Ali Ouni, Takashi Ishio, and Katsuro Inoue. 2018. Do Developers Update Their Library Dependencies? *Empirical Softw. Eng.* 23, 1 (2018), 384–417.
[47] H. Larsson, E. Lindqvist, and R. Torkar. 2014. Outliers and Replication in Software Engineering. In *Proc. Asia-Pacific Software Engineering Conf. (APSEC)*. IEEE CS, 207–214.
[48] Trang Le, Weixuan Fu, and Jason Moore. 2020. Scaling Tree-Based Automated Machine Learning to Biomedical Big Data with a Feature Set Selector. *Bioinformatics* 36, 1 (2020), 250–256.
[49] Y. Lee, N. Chen, and R. Johnson. 2013. Drag-and-Drop Refactoring: Intuitive and Efficient Program Transformation. In *Proc. Int'l Conf. Software Engineering (ICSE)*. IEEE CS, 23–32.
[50] Yun Lin, Xin Peng, Zhenchang Xing, Diwen Zheng, and Wenyun Zhao. 2015. Clone-Based and Interactive Recommendation for Modifying Pasted Code. In *Proc. Europ. Software Engineering Conf./Foundations of Software Engineering (ESEC/FSE)*. ACM, 520–531.

[51] Yun Lin, Jun Sun, Yinxing Xue, Yang Liu, and Jinsong Dong. 2017. Feedback-based Debugging. In *Proc. Int'l Conf. Software Engineering (ICSE)*. IEEE CS, 393–403.
[52] Jonathan Lung, Jorge Aranda, and Steve Easterbrook. 2008. On the Difficulty of Replicating Human Subjects Studies in Software Engineering. In *Proc. Int'l Conf. Software Engineering (ICSE)*. ACM, 191–200.
[53] Patrick Mäder and Alexander Egyed. 2015. Do Developers Benefit from Requirements Traceability when Evolving and Maintaining a Software System? *Empirical Softw. Eng.* 20, 2 (2015), 413–441.
[54] K McKiernan, J Kaufman, J. Kucera-Thompson, and J Binder. 2003. A Parametric Manipulation of factors Affecting Task-Induced Deactivation in Functional Neuroimaging. *J. Cognitive Neuroscience* 15, 3 (2003), 394–408.
[55] Helfried Moosbrugger and Augustin Kelava. 2007. *Testtheorie und Fragebogenkonstruktion*. Springer, Berlin.
[56] Randal S. Olson, Nathan Bartley, Ryan J. Urbanowicz, and Jason H. Moore. 2016. Evaluation of a Tree-based Pipeline Optimization Tool for Automating Data Science. In *Proc. Genetic and Evolutionary Computation Conference (GECCO)*. ACM, 485–492.
[57] Jongse Park, Hadi Esmaeilzadeh, Xin Zhang, Mayur Naik, and William Harris. 2015. Flexjava: Language Support for Safe and Modular Approximate Programming. In *Proc. ACM Joint Symposium Foundations of Software Engineering (FSE)*. 745–757.
[58] Norman Peitek, Janet Siegmund, Chris Parnin, Sven Apel, Johannes Hofmeister, and André Brechmann. 2018. Simultaneous Measurement of Program Comprehension with fMRI and Eye Tracking: A Case Study. In *Proc. Int'l Symposium Empirical Software Engineering and Measurement (ESEM)*. ACM, 24:1–24:10.
[59] Thomas Perneger. 1998. What's Wrong with Bonferroni Adjustments. *Bmj* 316, 7139 (1998), 1236–1238.
[60] Rolando P. Reyes, Oscar Dieste, Efraín R. Fonseca, and Natalia Juristo. 2018. Statistical Errors in Software Engineering Experiments: A Preliminary Literature Review. In *Proc. Int'l Conf. Software Engineering (ICSE)*. ACM, 1195–1206.
[61] Martin Schäf, Daniel Schwartz-Narbonne, and Thomas Wies. 2013. Explaining Inconsistent Code. In *Proc. Europ. Software Engineering Conf./Foundations of Software Engineering (ESEC/FSE)*. ACM, 521–531.
[62] Janet Siegmund, Christian Kästner, Sven Apel, Chris Parnin, Anja Bethmann, Thomas Leich, Gunter Saake, and AndrÃĺ Brechmann. 2014. Understanding Understanding Source Code with Functional Magnetic Resonance Imaging. In *Proc. Int'l Conf. Software Engineering (ICSE)*. ACM, 378–389.
[63] Janet Siegmund, Christian Kästner, Jörg Liebig, Sven Apel, and Stefan Hanenberg. 2014. Measuring and Modeling Programming Experience. *Empirical Softw. Eng.* 19, 5 (2014), 1299–1334.
[64] Janet Siegmund, Norman Peitek, Chris Parnin, Sven Apel, Johannes Hofmeister, Christian Kästner, Andrew Begel, Anja Bethmann, and André Brechmann. 2017. Measuring Neural Efficiency of Program Comprehension. In *Proc. Europ. Software Engineering Conf./Foundations of Software Engineering (ESEC/FSE)*. ACM, 140–150.
[65] Janet Siegmund, Norbert Siegmund, and Sven Apel. 2015. Views on Internal and External Validity in Empirical Software Engineering. In *Proc. Int'l Conf. Software Engineering (ICSE)*. IEEE CS, 9–19.
[66] Norbert Siegmund, Stefan Sobernig, and Sven Apel. 2017. Attributed Variability Models: Outside the Comfort Zone. In *Proc. ACM Joint Symposium Foundations of Software Engineering (FSE)*. ACM, 268–278.
[67] Dag Sjøberg, Bente Anda, Erik Arisholm, Tore Dyba, Magne Jorgensen, Amela Karahasanovic, Espen Frimann Koren, and Marek Vokác. 2002. Conducting Realistic Experiments in Software Engineering. In *Proc. Int'l Conf. Software Engineering (ICSE)*. IEEE CS, 17–26.
[68] Dag Sjøberg, Jo Hannay, Ove Hansen, Vigdis By Kampenes, Amela Karahasanovic, Nils-Kristian Liborg, and Anette Rekdal. 2005. A Survey of Controlled Experiments in Software Engineering. *IEEE Trans. Softw. Eng.* 31, 9 (2005), 733–753.
[69] Jean Talairach and Pierre Tournoux. 1988. *Co-Planar Stereotaxic Atlas of the Human Brain*. Thieme.
[70] Walter Tichy, Paul Lukowicz, Lutz Prechelt, and Ernst Heinz. 1995. Experimental Evaluation in Computer Science: A Quantitative Study. *J. Systems and Software* 28, 1 (1995), 9–18.
[71] Vladimir Vapnik and Alexey Chervonenkis. 2015. On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities. In *Measures of Complexity*. Springer, 11–30.
[72] Iris Vessey, Venkataraman Ramesh, and Robert Glass. 2002. Research in Information Systems: An Empirical Study of Diversity in the Discipline and Its Journals. *Journal of Management Information Systems* 19, 2 (2002), 129–174.
[73] Clifford Wagner. 1982. Simpson's Paradox in Real Life. *The American Statistician* 36, 1 (1982), 46–48.
[74] Zhaogui Xu, Shiqing Ma, Xiangyu Zhang, Shuofei Zhu, and Baowen Xu. 2018. Debugging with Intelligence via Probabilistic Inference. In *Proc. Int'l Conf. Software Engineering (ICSE)*. ACM, 1171–1181.
[75] K. Yskout, R. Scandariato, and W. Joosen. 2012. Does Organizing Security Patterns Focus Architectural Choices?. In *Proc. Int'l Conf. Software Engineering (ICSE)*. IEEE CS, 617–627.
[76] Koen Yskout, Riccardo Scandariato, and Wouter Joosen. 2015. Do Security Patterns Really Help Designers?. In *Proc. Int'l Conf. Software Engineering (ICSE)*. IEEE CS, 292–302.

# 9 APPENDIX

## 9.1 Differing Results: An Empirical Study on the Impact of Static Typing on Software Maintainability

Summary: The paper describes an experiment to evaluate how static and dynamic type systems affect the maintainability of software [32].

Independent variables:

(1) Type system (2 levels, operationalized with Java (static) and Groovy (dynamic))
(2) Tasks (9 levels, operationalized with 9 programming tasks of 3 different categories: Class-identification task (CIT, 5 tasks), type-error fixing task (TEFT, 2 tasks), semantic-error fixing task (SEFT, 2 tasks))

Dependent variable: Task completion time [metric scale]

Null hypotheses:

$H0_{CIT}$: Static type systems have no influence on development time for the task category *class identification*
$H0_{TEFT}$: Static type systems have no influence on development time task category *type-error fixing*
$H0_{SEFT}$: Static type systems have no influence on the debugging time for the task category *semantic-error fixing*

Results:

$H0_{CIT}$: Difference in favor of static type system for all but one class-identification task
$H0_{TEFT}$: Difference in favor of static type system for all type-error fixing task
$H0_{SEFT}$: No difference for the semantic-error fixing task

Although the authors defined different task categories, they did not do a per-category analysis, but a per-task analysis (and analysis of all tasks combined). This allowed them to disentangle the effect of specific tasks, but at the cost of statistical power. The authors did not adjust the $p$ level for multiple comparisons; with an FDR correction, we did not conclude a significant difference for CIT2 ($p$ value of 0.033). Next, we aggregated the data per task category, and then compared whether there is a difference within subjects between Round 1 and Round 2 regarding the development time, and also ran an ANOVA. We compare the data and significance of the original analysis and our aggregated analysis in Table 9. Regarding the within-subject comparison, we come to different conclusions than the authors. First, we do not replicate the rejection of $H0_{CIT}$, since the difference is significant for only one of the two groups in favor of the static type system (i.e., the group that started with the dynamic type system). For the group that started with the static type system, the difference is not significant. Different than the authors, who assume a favor for the static type system because of the difference for one group, we find this too liberal regarding the null hypothesis, such that we do not reject $H0_{CIT}$. For the remaining two hypotheses, we come to the same conclusion. Regarding the ANOVA, we come to the same conclusion of the authors, such that we have a significant main effect of the task category and a significant interaction effect of task category and type system in both rounds. We also found the significant main effect of the type system in the second round. To summarize, the aggregated reanalysis itself did not let us draw different conclusions than the authors. The difference regarding rejecting $H0_{CIT}$ is caused by a different interpretation of when the null hypothesis can be rejected or accepted.

Table 9. Average task completion times. The lower part of the table contains the aggregated times per task category. Highlighted cells indicate analysis we replicated of the authors with Wilcoxon tests (dark gray) or of the aggregated analysis with ANOVAs (light gray).

| | | CIT1 | CIT2 | CIT3 | CIT4 | CIT5 | TEFT1 | TEFT2 | SEFT1 | SEFT2 |
|---|---|---|---|---|---|---|---|---|---|---|
| Round 1 | Dynamic | 822.41 | 616.35 | 1073.71 | 992.18 | 1004.24 | 757.88 | 788.00 | 1022.35 | 538.59 |
| | Static | 762.06 | 1194.81 | 1069.88 | 1105.88 | 837.69 | 303.13 | 168.25 | 1613.69 | 794.44 |
| Round 2 | Dynamic | 812.44 | 725.81 | 1297.94 | 1062.31 | 1226.88 | 1108.88 | 914.31 | 591.63 | 311.94 |
| | Static | 321.65 | 391.35 | 570.47 | 564.82 | 552.00 | 172.53 | 126.82 | 638.82 | 237.35 |
| Within-Group Comparison (Dynamic first) | | V: 153, **p: <0.001** | V: 148, **p: <0.001** | V: 147, **p: <0.001** | V: 149, **p: <0.001** | V: 153, **p: <0.001** | V: 153, **p: <0.001** | V: 153, **p: <0.001** | V: 123, **p: 0.027** | V: 123, **p: 0.027** |
| Within-Group Comparison (Static first) | | V: 70, p: 0.94 | V: 109, p: 0.033 | V: 44, p:0.231 | V: 76, p: 0.706 | V: 11, **p: 0.002** | V: 2, **p: <0.001** | V: 1, **p: <0.001** | V: 132, **p: <0.001** | V: 125, **p: 0.002** |
| Round 1 | ANOVA | Type system: F: 0.089, p: 0.768, $\eta^2$ = 0.0005<br>Task: F: 11.402, p: <0.001, $\eta^2$ = 0.190<br>Type system x Task: F: 6.851, p: <0.001, $\eta^2$ = 0.114 | | | | | | | | |
| Round 2 | ANOVA | Type system: F: 42.54, p: <0.001, $\eta^2$ = 0.219<br>Task: F: 9.91, p: <0.001, $\eta^2$ = 0.130<br>Type system x Task: F: 6.45, p: <0.001, $\eta^2$ = 0.085 | | | | | | | | |
| Round 1 | Dynamic | 901.78 | | | | | 772.94 | | 780.47 | |
| | Static | 994.06 | | | | | 235.69 | | 1204.06 | |
| Round 2 | Dynamic | 1025.08 | | | | | 1011.59 | | 451.78 | |
| | Static | 480.06 | | | | | 149.68 | | 438.09 | |
| Round 1 | ANOVA | Type system: F: 0.007, p: 0.934, $\eta^2$ = 0.00006<br>Task: F: 26.28, p: <0.001, $\eta^2$ = 0.234<br>Type system x Task: F: 22.66, p: <0.001, $\eta^2$ = 0.202 | | | | | | | | |
| Round 2 | ANOVA | Type system: F: 27.83, p: <0.001, $\eta^2$ = 0.282<br>Task: F: 13.57, p: <0.001, $\eta^2$ = 0.076<br>Type system x Task: F: 27.45, p: <0.001, $\eta^2$ = 0.154 | | | | | | | | |
| Within-Group Comparison (Dynamic first) | | V: 153, **p: <0.001** | | | | | V: 153, **p: <0.001** | | V: 134, **p: 0.004** | |
| Within-Group Comparison (Static first) | | V: 57, p: 0.597 | | | | | V: 0, **p: <0.001** | | V: 135, **p: <0.001** | |

We could not replicate the exact values of the authors, but they do not deviate considerably.

## 9.2 Differing Results: Debugging with Intelligence via Probabilistic Inference

Summary: The authors developed an approach to include probabilistic information during the debugging process. Specifically, they describe the probability of how correct variables and statements are. They compare the approach to a standard debugger [74].

Independent variable: Debugging approach (2 levels, operationalized as the prototype of the authors and a standard debugger (pdb))
Tasks: Participants should solve a debugging tasks in 4 different programs (lcsubstr, quadratic, fibonacci, mergesort)
Dependent variable: Task completion time [metric scale]
Null hypothesis:
– There is no significant performance difference between the two groups
Results:
– The authors rejected the null hypothesis, such that their prototype reduced the debugging time for each task.

Table 10. Task completion time per task and participant. The authors ran a Wilcoxon test and we both replicated the Wilcoxon test as well as ran a $t$ test for task completion times. Highlighted cells indicate analysis we replicated of the authors (dark gray) or of the aggregated analysis (light gray).

| Participant | Task: lcsubstr | | Task: quadratic | | Task: fibonacci | | Task: mergesort | |
|---|---|---|---|---|---|---|---|---|
| | pdb | prototype | pdb | prototype | pdb | prototype | pdb | prototype |
| N1 / Y1 | 11.56 | 6.37 | 33.1 | 16.32 | 15.03 | 14.55 | 10.54 | 12.5 |
| N2 / Y2 | 30.06 | 8.2 | 10.14 | 11.22 | 27 | 16.11 | 18.13 | 12.23 |
| N3 / Y3 | 10.3 | 5.29 | 21.5 | 10.58 | 13.31 | 11.81 | 15.04 | 14.32 |
| N4 / Y4 | 5.8 | 5.5 | 19.31 | 17.76 | 17.24 | 18.33 | 18.33 | 13.9 |
| N5 / Y5 | 2.85 | 6.96 | 10.45 | 14.4 | 12.45 | 14.78 | 20.78 | 15.61 |
| N6 / Y6 | 21.11 | 4.89 | 12.8 | 10.33 | 25.69 | 15.6 | 20.5 | 13.8 |
| N7 / Y7 | 23.11 | 7.36 | 27.45 | 14 | 34.46 | 13.28 | 30.5 | 11.44 |
| N8 / Y8 | 12.71 | 8.11 | 20.45 | 15.4 | 18.51 | 18.44 | 14.33 | 14.49 |
| Mean | 14.69 | 6.59 | 19.40 | 13.75 | 20.46 | 15.36 | 18.52 | 13.54 |
| $p$ (Wilcoxon, original) | 0.006* | | 0.037* | | 0.049* | | 0.014* | |
| $p$ (Wilcoxon, reanalysis) | 0.011 | | 0.039 | | 0.098 | | 0.027 | |
| t (df) | 2.576 (7) | | 2.176 (7) | | 2.760 (7) | | 2.170 (7) | |
| $p$ (t test, reanalysis) | 0.018 | | 0.033 | | 0.061 | | 0.033 | |
| t (df) / $p$ (aggregated) | t = 4.489, df = 31, $p < 0.001$ | | | | | | | |

*These $p$ values cannot be replicated with a Wilcoxon test (see row $p$ (Wilcoxon, reanalysis)). The original $p$ values remain significant after FDR correction, but our computed $p$ values do not indicate a significant difference for any individual task.

We can replicate the results of the authors that adding probabilistic inference to describe the probability of how correct variables and statements are can make debugging faster. However, we found several inconsistencies in the analysis of the authors: First, there was no correction for multiple comparison, which, however, would not change the results. Second, and more importantly, we cannot replicate the $p$ values with the data provided in the

paper. We reran the one-sided paired Wilcoxon test with R[11], and we come to different $p$ values (provided in Table 10). With an FDR correction, a significant difference only for the first task remains, which is the task with the highest speed-up. When applying the full pipeline (i.e., check for normality, a t test (data for all tasks are normally distributed), and the FDR correction), no significant difference for an individual tasks remains. However, when aggregating the data for all four tasks together, we observe a highly significant effect. This highlights once again the importance of (a) applying the correct analysis pipeline, (b) describe it in detail, and (c) deciding an appropriate aggregation level is critical for interpreting data. In this case, we see results that could point in two directions: the prototype may or may not generally be more efficient for debugging than the standard debugger.

### 9.3 Different Result: Does Organizing Security Patterns Focus Architectural Choices?

Summary: Yskout and others conducted a large-scale study to analyze the effect of structuring security requirements on the time and efficiency to improve a given software architecture to meet a security requirement [75][12]

Independent variable: Type of requirement specification (2 levels, operationalized as structured and plain)
Tasks: 4 tasks to improve a given software architecture to meet a security requirement (plus a warm-up task that was not analyzed).
Dependent variables:

(1) Task completion time (referred to as "effort") [metric scale]
(2) Efficiency (operationalized as the number of security patterns implemented, divided by the total number of available security patterns) [metric scale]

Null hypotheses (for each of the four tasks):

$H_0 time$: The kind of requirement specification does not affect task completion time
$H_0 efficiency$: The kind of requirement specification does not affect efficiency

Results:

$H_0 time$: No significant difference for any of the tasks regarding completion time.
$H_0 efficiency$: Significant difference for 2 of the 4 tasks.

As aggregation function, we applied the arithmetic mean to both dependent variables (i.e., task completion time, efficiency). We aggregated over all tasks (excluding the warm-up task). Before that, we removed outliers as denoted by Yskout and others [75]. We applied the Wilcoxon test to both dependent variables, when the data were not normally distributed; otherwise we applied a t test for stronger statistical power. We summarize our reanalysis and which test we applied in Table 11. For the task completion time, we could replicate the result of no significant difference, and come to the same conclusion as the authors. For efficiency, the aggregation yielded a significant difference, so we also reject the null hypothesis.

In general, we can confirm the results of the authors. However, we like to note that in the introduction and discussion of results, the authors were not very precise when discussing the benefit for efficiency. That is, although they defined one null hypothesis for each task, they did not restrict their discussion to the tasks for which there was a difference in efficiency, but kept the discussion very general, saying that they could find a benefit for efficiency. This can easily be interpreted that, independent of the task, a structured requirement specification leads to a higher efficiency, which would be a too strong statement. Instead, the authors should have discussed the results task-wise, allowing them a more nuanced discussion, especially since there are two tasks that seem to be better solvable with the structured requirement specification.

[11]Unfortunately, the authors did not specify the details, so we used the test that comes closest to the authors' $p$ values. Given the hypothesis of the authors, we should have actually used a two-sided version.
[12]Link to data: https://distrinet.cs.kuleuven.be/software/securitypatterns/

Table 11. Task completion times and efficiency for each of the tasks. We ran a Wilcoxon test when data were not normally distributed, otherwise a t test. Highlighted cells contain values that are computed by us.

| Task | Kind of specification | Time | Efficiency |
|---|---|---|---|
| B | Plain | 1639 | 0.139 |
| | Structured | 1614 | 0.349 |
| | W | 0.670 | 3.891 |
| | $p$ value | 0.503 | <0.001 |
| C | Plain | 1220 | 0.109 |
| | Structured | 1648 | 0.239 |
| | W/t | 1.282 | 3.884 |
| | $p$ value | 0.207 | 0.0001 |
| D | Plain | 1911 | 0.139 |
| | Structured | 2109 | 0.146 |
| | t | 0.582 | 0.003 |
| | $p$ value | 0.564 | 0.998 |
| E | Plain | 1373 | 0.185 |
| | Structured | 1701 | 0.111 |
| | W | 1.252 | 1.673 |
| | $p$ value | 0.21 | 0.094 |
| Aggregated | Plain | 1763.53 | 0.21 |
| | Structured | 1483.81 | 0.14 |
| | W | 1.15 | 2.85 |
| | $p$ value | 0.25 | 0.004 |

## 9.4 Differing Results: Do Security Patterns Really Help Designers?

This study is by the same author team of the Study "Does Organizing Security Patterns Focus Architectural Choices?" ([75]) and has a similar structure [76]. The authors conduct a similar study, in which 64 graduate students in pairs of two should ensure the security of a banking system by completing six tasks. In a nutshell, the authors could not find an effect of security patterns, neither on time to solve task nor on the number of covered misuse cases. However, they found that some tasks were solved more often correctly when security patterns were provided.

Independent variable: Security patterns (2 levels, operationalized as present or not present
Tasks: 6 different tasks to improve security of a given software system (plus a warm-up task that was not analyzed
Dependent variables:

(1) Task completion time [metric scale]
(2) Number of covered misuse cases (1 to 5) [ordinal scale]
(3) Correctness (wrong, some errors, correct) [ordinal scale]

Null hypotheses:

(1) The usage of security patterns has no influence on the mean time needed to complete a task.

Table 12. Completion time, number of covered misuse cases, and correctness for each task, as well as for all tasks. Highlighted cells indicate analysis we replicated with Wilcoxon tests of the authors (dark gray) or of the aggregated analysis (light gray).

| | | Tasks | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Variable | Pattern? | B | C | D | E | F | G | All |
| Time [h] | Not present | 0.78 | 0.89 | 0.88 | 0.81 | 0.91 | 0.53 | 2.42 |
| | Present | 0.92 | 1.06 | 1.09 | 0.87 | 1.02 | 0.69 | 2.81 |
| U/t | | 0.418 | 0.302 | 0.661 | 0.036 | 1.481 | 0.245 | 1.242 (df = 30) |
| $p$ value | | 0.676 | 0.763 | 0.509 | 0.970 | 0.138 | 0.806 | 0.219 |
| Misuse cases | Not present | 43 | 13 | 31 | 48 | 38 | 39 | 212 |
| | Present | 65 | 20 | 34 | 50 | 35 | 53 | 257 |
| U | | 1.557 | 1.150 | 0.925 | 0.188 | 0.589 | 0.944 | 1.866 |
| $p$ value | | 0.119 | 0.250 | 0.355 | 0.851 | 0.556 | 0.345 | 0.062 |
| Correctness | Not present | 18 | 5 | 20 | 20 | 17 | 16 | 96 |
| | Present | 25 | 17 | 21 | 26 | 17 | 25 | 131 |
| U | | 0.361 | 2.977 | 0.850 | 1.809 | 1.102 | 1.529 | 3.525 |
| $p$ value | | 0.718 | **0.003** | 0.396 | 0.070 | 0.271 | 0.126 | 0.0004 |

(2) The usage of security patterns has no influence on the mean number of covered misuse cases for a task.[13]
(3) The usage of security patterns has no influence on the correctness scores of a task (not explicitly stated in the paper, but still tested).

Results:

(1) No significant effect of security patterns on the time to solve task
(2) No significant effect of security patterns on the number of covered misuse cases
(3) A significant effect for two of the six tasks for correctness

We summarize the number of covered misuse cases, correctness, and time to solve the tasks in Table 12. We cannot replicate the $p$ values with Wilcoxon tests (data were not normally distributed) that the authors reported, possibly because we used Python (scipy) for the reanalysis, while the authors used R. This leads to the fact that we did not observe a significant difference in correctness for Task E, although the authors did. However, the authors did not correct for multiple comparisons. Using an FDR correction based on the $p$ values of the authors, the difference for Task E also vanishes.

In the reanalysis, we aggregated the data over all tasks, such that we sum up the values for all tasks and then compare the means (task completion time) and ranks (number of misuse cases, correctness). In essence, we can confirm that for response time and number of misuse cases, the presence of security patterns has no effect. However, for correctness, we find a significant difference over all tasks in favor of security patterns.

To conclude, for this case, the aggregation did not really change the overall picture. We confirmed the authors' result that correctness can be affected by the presence of security patterns. With the task-wise analysis, the authors looked deeper into the data. What would have been interesting now is to take a closer look at how Task E differs from the others. For example, this task was the only task for which the security pattern *Demilitarized Zone* was relevant. Maybe this pattern is especially helpful to understand how to implement security updates to an existing application (in this case, to prevent read access).

[13]Technically, the mean is not correct here, as the authors used a Wilcoxon test, which does not compare the means, but the rank sums.

## 9.5 Same Results: Boa: A Language and Infrastructure for Analyzing Ultra-Large-Scale Software Repositories

Summary: The authors developed a domain specific language called Boa to ease the process of querying information from many open-source projects (e.g., from SourceForge or GitHub). They evaluated Boa compared to Java by letting participants write scripts in each language. The scripts were analyzed regarding lines of code and execution times.

Independent variable: Language (2 levels, operationalized as Boa and Java)
Tasks: 21 typical mining tasks of four different areas:
- A Programming language (3 tasks, A1 to A3)
- B Project management (11 tasks, B1 to B11)
- C Legal (2 tasks, C1 and C2)
- D Platform/environment (5 tasks, D1 to D5)

in three categories:
- Metadata only (Tasks A1, A2, B1, B2, C1, C2, D1 to D5)
- Data from one or few revision (Tasks A3, B3 to B5)
- Data from most of the revisions (Tasks B6 to B11)

Dependent variables:
(1) Lines of code per language and task [metric scale]
(2) Execution time per language (Java: repositories cached) and task [metric scale]
(3) Execution time per language (Java: repositories remotely accessed) and task [metric scale]

Results:
(1) 8 to 18 fewer lines of Boa code compared to Java code
(2) 8 to 250 times of speed up for Boa code compared to Java code (repositories cached)
(3) 459 to 2364 times of speed up for Boa code compared to Java code (repositories remotely accessed)

Participants created one script for each task, so for each dependent variable, there was one data point (e.g., 89 Lines of code for Task A3 for the Java script). Hence, it was not possible to conduct a significance test or state a null hypotheses. Instead, the authors reported the differences in terms of multiples (e.g., the Java script consist of 9x as much lines of code as the Boa script). In Table 13, we show the according data. In their discussion, the authors highlighted the maximum benefit, especially when comparing the remote access of Java scripts to the Boa scripts, they speak of an improvement of more than 2 000 times. Aggregating the data according to the three categories of the tasks defined by the authors, this number reduces to 1 800, and aggregating further, it becomes 1 300, which still can be seen as considerable improvement, without highlighting the maximum benefit. Thus, in this case, aggregation would not fundamentally alter the conclusions that can be drawn (i.e., that Boa scripts are shorter and faster), but the numbers reflect the average case, instead of the maximum case. Especially when it comes to evaluating a newly developed approach, it is important to be more objective and consider the aggregated data, instead of promoting the best values.

The authors also analyzed the scalability of the approaches, which also happened task-wise. In Figure 3, we show the original figure (Fig. 15 in the original paper), and additionally the aggregated version. Again, the information point to the same conclusion, that is, that for 60k projects and upwards, Boa is faster. The non-aggregated plot shows the additional information that this is true for all tasks, and not, for example, better for 80 % of the tasks and considerable worse for 20 % of the tasks. By contrast, the aggregated plot shows the data in a more concise way. Thus, depending on the underlying data, one could decide whether to aggregate or not to aggregate the data. In this case, the aggregated plot does not hide any irregularities in the data, so it might be a good choice.

Table 13. Raw data for evaluation of Boa. Gray cells contain values computed by us.

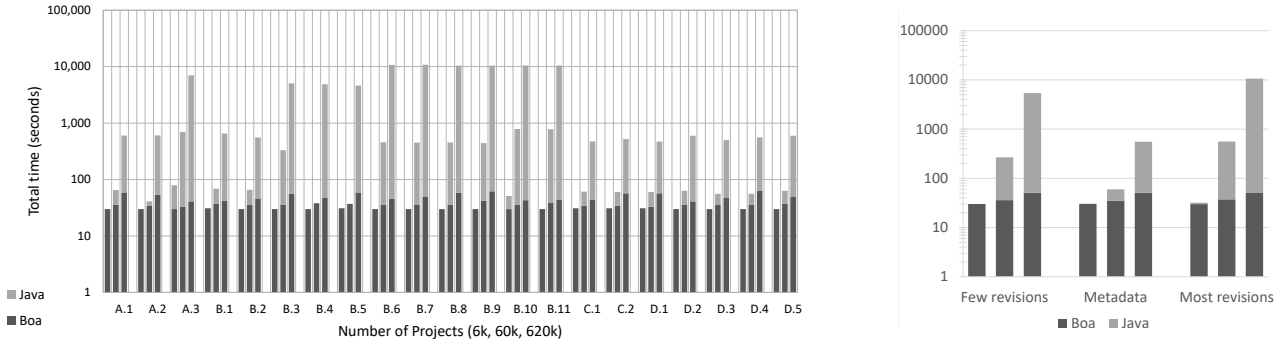
| Category | Task | LOC Java | LOC Boa | LOC Diff | Time Java (cached) | Time Boa | Time Diff (cached) | Time Java (remote) | Time Diff (remote) |
|---|---|---|---|---|---|---|---|---|---|
| Metadata | A1 | 61 | 4 | 15.25 | 602 | 59 | 10.20 | | |
| | A2 | 32 | 4 | 8.00 | 603 | 51 | 11.82 | | |
| | B1 | 43 | 3 | 14.33 | 651 | 42 | 15.50 | | |
| | B2 | 45 | 4 | 11.25 | 556 | 46 | 12.09 | | |
| | C1 | 63 | 4 | 15.75 | 474 | 44 | 10.77 | | |
| | C2 | 32 | 4 | 8.00 | 522 | 57 | 9.16 | | |
| | D1 | 61 | 4 | 15.25 | 469 | 57 | 8.23 | | |
| | D2 | 33 | 4 | 8.25 | 597 | 41 | 14.56 | | |
| | D3 | 61 | 4 | 15.20 | 498 | 47 | 10.60 | | |
| | D4 | 32 | 4 | 8.00 | 558 | 64 | 8.72 | | |
| | D5 | 71 | 5 | 14.20 | 598 | 49 | 12.20 | | |
| One/few revisions | A3 | 89 | 10 | 8.90 | 6998 | 41 | 170.68 | 45793 | 1116.90 |
| | B3 | 66 | 6 | 11.00 | 5053 | 56 | 90.23 | 25690 | 458.75 |
| | B4 | 107 | 6 | 17.83 | 4880 | 48 | 101.67 | 18700 | 389.58 |
| | B5 | 60 | 5 | 12.00 | 4636 | 59 | 78.58 | 17888 | 303.19 |
| Most revisions | B6 | 76 | 6 | 12.67 | 10750 | 45 | 238.89 | 95404 | 2120.09 |
| | B7 | 69 | 6 | 11.50 | 10821 | 50 | 216.42 | 85265 | 1705.30 |
| | B8 | 72 | 4 | 18.00 | 10435 | 58 | 179.91 | 95755 | 1650.95 |
| | B9 | 68 | 5 | 13.60 | 10431 | 62 | 168.24 | 88440 | 1426.45 |
| | B10 | 79 | 6 | 13.17 | 10489 | 43 | 243.93 | 100883 | 2346.12 |
| | B11 | 82 | 6 | 13.67 | 10518 | 44 | 239.05 | 88279 | 2006.34 |
| Metadata | | 48.55 | 4.00 | 12.14 | 557.09 | 50.64 | 11.00 | | |
| One/few revisions | | 80.5 | 6.75 | 11.93 | 5391.75 | 51.00 | 105.72 | 27017.75 | 529.76 |
| Most revisions | | 74.33 | 5.5 | 13.52 | 10574.00 | 50.33 | 210.08 | 92337.67 | 1834.52 |
| All | | 62 | 4.95 | 12.52 | 4339.95 | 50.62 | 85.74 | 66209.70* | 1308.49* |

*Only based on the categories *one/few revisions* and *most revisions*

## 9.6 Same Results: Explaining Inconsistent Code

Summary: The authors developed an approach to automatically detect and highlight inconsistent code via the construction of error invariant automata (EIA). In a small user study, they found that their approach helped participants to detect inconsistencies in code [61].

Independent variable: Approach to display inconsistent code (2 levels, operationalized with visual assistance based on invariant automata, and without visual assistance)
Tasks: 6 tasks to detect inconsistencies in different code snippets
Dependent variable: Task completion time [metric scale]
Null hypotheses:
No test was conducted because of small sample size (12 participants), so no hypotheses were stated.




(a) Unaggregated data from the original study.

(b) Aggregated data from our reanalysis

Fig. 3. Task completion times for Boa, compared to Java, for different project sizes.

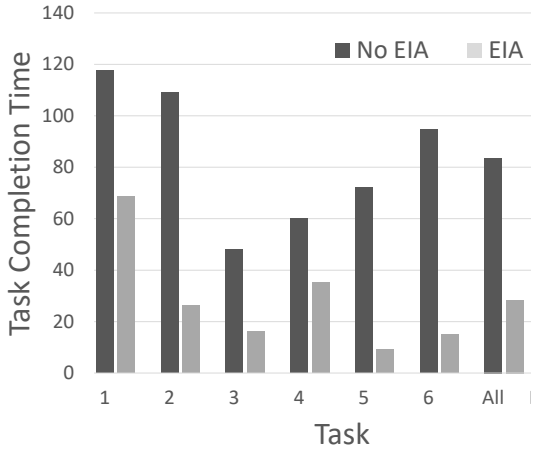



Fig. 4. Task completion time to detect inconsistencies in code. EIA: Error Invariant Automata (author's approach).

Results: Speed up by a factor of three for the version that created visual assistance based on error invariant automata.

We agree with the authors that with such a small sample size, a statistical test does not make much sense. However, when aggregating the response times for all tasks, we can increase statistical power, so that we can conduct a significance test here. Since the data are not normally distributed, we use a paired Wilcoxon test, showing that the speed up in favor of the visual assistance based on error invariant automata is significant ($W = 75, p = 0.002$). In Figure 4, we show the task-wise response times, as well as the aggregated response times over all tasks (right in Figure 4). Thus, in this case, aggregation did not lead to different results, but we can state more determined that using the error invariant automata to visualize inconsistencies in code does significantly help participants to detect inconsistencies.

## 9.7 Same Results: Feedback-Based Debugging

Summary: The authors present their approach to integrate light-weight feedback in the debugging process, and based on this, automatically recommend suspicious execution traces. They implement their approach in the tool Microbat, and compare it to the Whyline tool. They found that participants who used Microbat are significantly faster in debugging [51].

Independent variable: Tool (2 levels, operationalized with Microbat and Whyline)
Tasks: 3 debugging tasks on three different programs
Dependent variable: Task completion time (referred to as performance) [metric scale]
Null hypothesis:
– No difference in performance between the two groups for none of the tasks
Results:
– The authors reject all three null hypotheses, such that Microbat reduces the task completion time for all tasks.

In the reanalysis (cf. Table 14), we aggregated the data over all three tasks, and we come to the same conclusion as the authors: The Wilcoxon test showed that over all tasks, the participants who used Microbat were significantly faster than the participants who used Whyline. We used the Wilcoxon test (not the t test), since the Shapiro-Wilk test revealed a deviation from normality. The authors did not correct the $p$ level for multiple comparison, but an FDR correction does not lead to different conclusions.

## 9.8 Same Results: FlexJava: Language Support for Safe and Modular Approximate Programming

Summary: The authors develop FlexJava, an approach to support approximate programming for the Java programming language. They compare it to EnerJ as baseline [57].

Independent variable: Tool support (2 levels, operationalized with, FlexJava and EnerJ)
Tasks: 3 different benchmarks, each with two versions in different orders
Dependent variables: Task completion time [metric scale]
Null hypotheses:
Not explicitly stated, but to compare the time to annotate source code with each of the two tools
Results:
(1) Authors state that there is a significant difference, yet did not report on a significance test

Our reanalysis consisted of two steps. First, we conducted the significance tests. To this end, we first checked for normality and then applied the appropriate test (t test for dependent samples or Wilcoxon). In Table 15, we summarize the results of our reanalysis.

For this study, it does not make a difference whether we aggregate the data or not; the difference is significant on the task level as well as over all tasks. Thus, we have provided evidence for the authors' statement of a significant difference. To reanalyze the data, we have extracted the numbers from Figure 9 of the paper, as we could not access the exact numbers. Although the authors provide a replication package, which is approved by the Replication Packages Evaluation Committee of FSE 2015, we could not find the results of the study in the repository.

Table 14. Task completion times per task and participant. The authors ran Wilcoxon tests. We ran Wilcoxon tests when data were not normally distributed, otherwise t tests. Highlighted cells indicate analysis we replicated of the authors (dark gray) or of the aggregated analysis (light gray).

| Group | Participant | Task 1 | Task 2 | Task 3 |
|---|---|---|---|---|
| Microbat | P1 | 5.7 | 8.1 | 10 |
| | P2 | 10 | 9.8 | 7.8 |
| | P3 | 9.7 | 4.2 | 7 |
| | P4 | 12.1 | 9.5 | 10.5 |
| | P5 | 20.4 | 7.3 | 13.5 |
| | P6 | 16 | 11.4 | 6.5 |
| | P7 | 12.2 | 10.7 | 11.2 |
| | P8 | 33.2 | 22.9 | 12.6 |
| Whyline | P9 | 15.5 | 18 | 12.1 |
| | P10 | 10.2 | 25.5 | 25.4 |
| | P11 | 25.5 | 10.1 | 19.5 |
| | P12 | 36.2 | 32.7 | 25.1 |
| | P13 | 35.2 | 35.1 | 35.3 |
| | P14 | 42.3 | 34.8 | 13 |
| | P15 | 27.2 | 47.4 | 22.5 |
| | P16 | 48.6 | 39.5 | 43.4 |
| Microbat | Average | 14.9 | 10.5 | 9.9 |
| Whyline | | 30.1 | 30.4 | 24.5 |
| | $p$ value (original) | 0.012* | 0.012* | 0.012* |
| | W / t (df) | 2.746 | 2.94 (14) | 3.790 |
| | $p$ value (reanalysis) | 0.0158 | 0.003 | 0.002 |
| Microbat | Average | | 11.76 | |
| Whyline | | | 28.34 | |
| | W | | -4.815 | |
| | $p$ value (reanalysis) | | < 0.001 | |

* We cannot replicate the exact $p$ values.

Table 15. Task completion times for each task. We ran a t test for the "sor" approach and Wilcoxon for the other two tasks. Gray cells contain values computed by us.

| Approach | sor | smm | fft | mean |
|---|---|---|---|---|
| FlexJava | 2.8 | 1.95 | 3 | 2.583 |
| EnerJ | 14.9 | 18.44 | 19.94 | 17.76 |
| t / W | -9.883 | 0 | 0 | -8.908 |
| df | 9 | – | – | 9 |
| $p$ | <0.001 | 0.002 | 0.006 | <0.001 |