How (Not) To Write a Software Engineering Abstract

Lutz Prechelt, Lloyd Montgomery, Julian Frattini, Franz Zieris

Abstract—*Background:* Abstracts are a particularly valuable element in a software engineering research article. However, not all abstracts are as informative as they could be. *Objective:* Characterize the structure of abstracts in high-quality software engineering venues. Observe and quantify deficiencies. Suggest guidelines for writing informative abstracts. *Methods:* Use qualitative open coding to derive concepts that explain relevant properties of abstracts. Identify the archetypical structure of abstracts. Use quantitative content analysis to objectively characterize abstract structure of a sample of 362 abstracts from five presumably high-quality venues. Use exploratory data analysis to find recurring issues in abstracts. Compare the archetypical structure to actual structures. Infer guidelines for producing informative abstracts. *Results:* Only 29% of the sampled abstracts are *complete*, i.e., provide background, objective, method, result, and conclusion information. For structured abstracts, the ratio is twice as big. Only 4% of the abstracts are *proper*, i.e., they also have good readability (Flesch-Kincaid score) and have no informativeness gaps, understandability gaps, nor highly ambiguous sentences. *Conclusions:* (1) Even in top venues, a large majority of abstracts are far from ideal. (2) Structured abstracts tend to be better than unstructured ones. (3) Artifact-centric works need a different structured format. (4) The community should start requiring conclusions that generalize, which currently are often missing in abstracts.

Index Terms—qualitative analysis, quantitative analysis, guidelines

1 Introduction

Although the abstract is a super important part of any research article [1], [2], when reading an abstract in software engineering — even in a presumably top-quality venue — we often feel it is lacking important information or find it difficult to understand at all.

We aim to substantiate this impression by operationalizing *abstracts quality* and analyzing hundreds of abstracts. We want to formulate constructive advice for writing better abstracts.

1.1 Research Questions

We ask several questions:

RQ1 What does a typical well-written abstract look like?

RQ2 **a)** Which deficiencies occur? **b)** How often?

RQ3 Do structured abstracts have better quality than unstructured ones?

RQ4 How *should* software engineering abstracts be written?

We consider RQ1 and RQ2 to be exploratory. RQ3 is hypothesis-driven; we expect a 'yes'. The answer to RQ4 will be derived from the answers to the other three.

1.2 Research Approach

No firm expectations regarding RQ1 and RQ2a exist for engineering articles, so qualitative methods will have to be used for them: We start our research with *open coding* of software engineering abstracts to derive a vocabulary (a set of concepts or *codes*; [3, Ch. 5]) by which the nature of a particular abstract can be characterized.

We intend to convince even readers who are skeptical of qualitative research regarding our answers to the research questions and the need to improve the quality of software engineering abstracts. We therefore perform a repeatable quantitative content analysis [4, Ch. 7] on a large sample of 362 presumably high-quality abstracts. We apply an elaborate eight-step approach for maximizing reliability.

The final stage is a statistical evaluation of the content analysis data, which is again exploratory and straightforward in the questions it asks and the statistical methods it applies.

1.3 Research Contributions

Our contributions correspond to the research questions as follows:

RQ1 As for the structure of well-written abstracts, we present an "abstracts archetype" that describes fixed parts of the structure and degrees of freedom (Section 4.3).

RQ2 We describe and discuss eight types of deficiencies; we quantify the frequency of each deficiency type for the entire sample of abstracts as well as for different subgroups of interest (Section 4 and Table 2).

RQ3 We present convincing data that structured abstracts tend to be better in several respects (Sections 4.5 & 4.13).

RQ4 We provide data-based how-to instructions for writing abstracts for authors; we provide guidance for editors and conference organizers (Section 6.2). Software engineering works should use structured abstracts, but need a different and more flexible template than what is used so far.

L. Prechelt is with Freie Universität Berlin, Germany
L. Montgomery is with University of Hamburg, Germany
J. Frattini is with Chalmers University of Technology and University of Gothenburg, Sweden
F. Zieris is with Blekinge Institute of Technology (BTH), Karlskrona, Sweden

2 Related Work

There is considerable literature on research abstracts across disciplines. We will not attempt to summarize it here, but provide examples of the different major perspectives of those studies. We otherwise focus on what has been done in the software engineering domain.

2.1 Abstracts Structure

Swales [5] introduced *genre analysis* as a means for teaching academic reading and writing, especially to non-native speakers: Genres are "classes of communicative events" (e.g., *the writing and reading of abstracts*) that are owned by a "discourse community" (e.g., *software engineering researchers*); genre analysis means deconstructing texts (from a genre) to better understand their elements in terms of their syntactical structure, content, role, and interrelationships (e.g., their relative position within the whole).

For our purposes here, the most relevant idea from genre analysis is the notion of "moves", which are, roughly speaking, the building blocks used by writers for making their overall point. Several studies have looked at the move structure of research abstracts in different fields such as applied linguistics [6] or protozoology [7]. Despite the differences of research content, they find very similar moves, typically the following five-move structure [7]:

> (1) situate the research within the scientific community; (2) introduce the research by describing the main features or presenting its purpose; (3) describe the methodology; (4) state the results; (5) draw conclusions or suggest practical applications.

This reflects, in slightly extended form, the IMRAD structure (Introduction, Methods, Results, and Discussion) of the body of scientific articles that has gradually become the norm since the 1940s [8].

We found a similar structure for abstracts of empirical works in software engineering (see Section 4.3), but abstracts of artifact-centric works (tool building) do not fit this model and need an extended one 4.6.3.

2.2 Abstracts Quality – What is "good"?

Several (meta-)studies on abstracts focus on quality assessment, most often in subfields of the biomedical domain. Many such studies cover articles of a homogeneous nature: randomized controlled trials (controlled experiments). This allows formulating specific expectations regarding what information should be presented in an abstract and allows performing the analysis in checklist fashion, for example: In clinical dermatology, [9] used a 30-item checklist on 197 abstracts for computing a 0-to-1 completeness score and found mean scores between 0.64 and 0.78 for their various subgroups. In dental medicine, [10] used a 29-item checklist on 100 abstracts and found a mean score of only 0.54. Among 303 abstracts of cost-effectiveness analyses, 29% did not report the baseline to which the intervention had been compared [11]. Among 146 abstracts of meta-analyses in peridontology, 33% did not even report the direction in which the evidence was pointing [12].

Various studies have investigated "spin" in the context of significance testing. The term covers two types of behavior: Using language that sounds more positive than warranted or reporting a secondary or alternative statistic as if it was the main one of interest. Among all abstracts reporting non-significant results, studies found spin in 45% of abstracts of orthopedic controlled experiments [13], 44% in emergency medicine [14], 56% in psychiatry and psychology [15], and 58% for the conclusions alone across a broad set of medical controlled experiments [16].

Unfortunately, the methods of those meta-studies are not applicable to a broad sample of software engineering abstracts, because in fields with heterogeneous study structures such as ours, the operationalization of *quality* is less straightforward. For example, *spin* can take many more forms in software engineering articles than is assumed by the studies mentioned above. It is difficult to decide which forms are acceptable and which are not.

One approach for discussing the quality of a software engineering abstract could be through comparing to a known "good" structure for abstracts. For instance, [7] remarks that one third of the 12 analyzed abstracts is lacking move 2 (stating a purpose).

2.3 Structured vs. Unstructured Abstracts

Many studies of abstracts quality do not study quality in general. For instance, neither [9] nor [10] reports which of their checklist items are missing most frequently. Rather, their research question is the relative quality of structured versus unstructured abstracts. Definition: A *structured abstract* is one that uses a prescribed sequence of intermediate headings, such as Background, Objective, Methods, Results, Conclusions or some similar set. In almost all of those meta-studies, including [9] and [10], the answer is: structured abstracts have fewer quality issues. Such research can be highly influential. For instance, the CONSORTS report [17] (containing guidelines for reporting controlled experiments, with over 10,000 citations), relies on such a study [18] to recommend structured abstracts.

In software engineering, Kitchenham proposed *Evidence Based Software Engineering* (EBSE) in 2004 [19]. EBSE relies a lot on Systematic Literature Reviews (SLR). The practicality of SLRs hinges on the informativeness of abstracts: Can the researcher decide quickly and reliably, whether the present article belongs in the SLR or not?

Therefore, Kitchenham performed two studies on structured abstracts in software engineering. The first took 23 published non-structured abstracts, converted them into structured ones, and compared the two versions. It found that the structured abstracts were much longer, but also had much better readability scores [20]. The second, by Budgen, Kitchenham, and others [21], is a controlled experiment based on similar pairs of abstracts rewritten into the structure Background, Aim, Methods, Results, Conclusions. 20 students and 44 researchers and practitioners each judge one structured and one different unstructured abstract for completeness (using an 18-item checklist) and clarity (using a vague 1-to-10 scale). The structured format was found to increase the completeness score by 6.6 and the clarity score by 3.0. 70% of the subjects also preferred the structured format subjectively. Both studies use only abstracts of

purely empirical studies, not tool-building works, for which structured abstracts require and extended form as we will see in Section 4.6.3.

3 Methods

3.1 Overview

Our study is a full-blown content analysis in the sense of Krippendorff [4]: not just a counting exercise with a fixed codebook, but rather an iterative codebook development before (and during) the counting and an extensive abductive inference exercise after the counting. It can be conceptualized as consisting of four widely overlapping stages or phases as shown in Figure 1: Codebook development, which

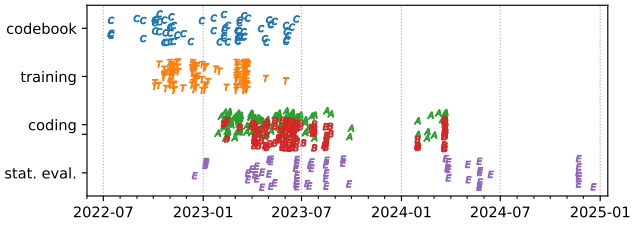

Fig. 1. Timeline of the main study phases and their individual events. Each character's x-coordinate represents the time of a git commit. The vertical scattering is added for legibility only.

started first and is described in Sections 3.3 and 3.4, defined the rules and target concepts of the counting. Training (Section 3.5) was for developing a joint understanding of the codebook. It also contributed greatly to the codebook's early evolution. Coding worked on a large sample of abstracts from top-quality venues described in Section 3.6 and produced the count data subsequently used in the statistical analysis. Our coding process and our subsequent statistical evaluation are described in Sections 3.7 and 3.8.

Overall, we consider our study to be a *qualitative* one, but coding and statistical evaluation are also largely compatible with a *positivist* epistemology (aiming for "objective" results), such that the final interpretation is done on a solid *quantitative* foundation.

3.2 Data Availability

We publish not only the outcome of our study, but also most parts of its development and execution history in full detail: as a git version repository. It includes all versions of the codebook, handling procedure, coded abstracts, Python scripts for automation, Python scripts for tabulations and plots, and the manuscript of this article in Knitr LaTeX format (that is, with automated computation of most of the numbers in the text). Find it at https://github.com/serqco/qabstracts/. A snapshot of the most important files is available at Zenodo as [22].

When we refer to abstracts from our sample, we use abbreviations of the first three authors' names and the year of publication, e.g., [BesMarBos22]. Such strings are hyperlinks to the annotated abstract in our GitHub repository. Refer to the repository in order to find the corresponding bibliographic information.

3.3 General Coding Rules

Besides the definition of the content categories (codes), our codebook contains global rules for the coding that can be summarized as follows:

- In order to limit complexity and avoid arbitrariness, we code by sentence and prefer single codes per sentence over multi-codings.
- In order to mimic ordinary readers, when choosing a code, we consider only what we have seen before plus one sentence forward context when needed (and only when needed).
- In order to avoid excessive criticism of abstracts quality, we avoid coding negative properties whenever an alterative, more positive interpretation is plausible as well.

3.4 Codebook and Codebook Development

3.4.1 BACKGROUND, OBJECTIVE, METHOD, RESULT, CONCLUSION

The codebook was initialized with concepts for the five sections commonly used in a structured abstract:[1] BACKGROUND, OBJECTIVE, METHOD, RESULT, CONCLUSION. These represent, respectively: context information, the study goal or question, empirical approach, empirical outcomes, and a take-home message that generalizes beyond the results.

Each code in the codebook is defined by a short verbal explanation. For example, we defined METHOD as *"information about the approach or setup of an empirical (or possibly purely mathematical) study"*. The initial definitions were made more and more precise and unambiguous by later codebook refinements. The part *"(or possibly purely mathematical)"* is such a clarification that we added when we encountered the first of those (very few) purely mathematical studies during the coding process and were confused which code was appropriate. To help disambiguation, a few of the definitions need to be much longer than the above.

3.4.2 *Artifact-centric Studies:* DESIGN

We then had a phase (internally called "prestudy") when we refined and extended the codebook based on coding attempts for a stratified sample of 20 abstracts from ICSE 2021. We quickly recognized that additional codes were needed.[2] Most importantly, many software engineering articles do not talk about only an empirical study. Rather, their focus is the design of some artifact, most often a tool, sometimes a method or something else. Much of the abstract is then spent on design considerations, design decisions, techniques applied in implementation, and so on. Such artifact-centric studies, although they also usually also contain an empirical study, are quite different in nature from purely empirical

1. In Krippendorff's terminology, this is an "established theories" justification of analytical constructs [4, Section 9.2.3].

2. In Krippendorff's terminology, this is an "expert knowledge and experience" justification of analytical constructs [4, Section 9.2.2]: Being software engineering researchers ourselves, we recognize when a sentence makes a different kind of contribution to an abstract than can be described by existing codes, and we are able to define what kind of contribution it is.

studies; we therefore introduced the code DESIGN to mark such material.[3]

We call the articles that contain at least one DESIGN coding in their abstract "design articles", the others "empirical articles".

3.4.3 *Refinements:* GAP, SUMMARY, FPOSS, *etc.*

At various later points during our study, we recognized a need for more granular coding in order to capture differences in abstract writing we wanted to measure. This led to the splitting of existing codes and the introduction of additional ones. We then reworked existing codings to use the new codes consistently throughout.

The most important cases of such additional codes are these: GAP states what is unknown or not yet possible; SUMMARY summarizes several results, but does not provide new information (whereas CONCLUSION generalizes beyond the immediate results); FPOSS and FNEED sentences occur at the end of abstracts — they state what future work is now possible or needed.

3.4.4 *Codes for Announcements:* A-*

Sometimes, a statement in an abstract insinuates there will be certain information in the article body, but does not provide any concrete information itself. We call such statements "announcements" and use codes with an A-* prefix for them. For example, here is an A-METHOD (method announcement) from [BesMarBos22]: *"As a second step, this study sets out to specifically provide a detailed assessment of additional and in-depth analysis of technical debt management strategies based on an encouraging mindset and attitude from both managers and technical roles to understand how, when and by whom such strategies are adopted in practice."* Despite the long sentence, we learn nothing about how the assessment or the analysis work. Here is an A-RESULT (results announcement) from [FlyChaDye22]: *"Based on those results, we then used the Boa and Software Heritage infrastructures to help identify and quantify several sources of dirty Git timestamp data."* The first part is METHOD, the second should have been RESULT, but, alas, we learn nothing about those sources' nature or number or impact.

3.4.5 *Codes for Headings:* H-*

We use H-* codes for the headings used in structured abstracts. These codes are conceptual, i.e., for instance *"Aim:"*, *"Goal:"*, *"Objective:"*, *"Question:"*, and their plural forms would all be coded as H-OBJECTIVE. Likewise, there are H-BACKGROUND, H-METHOD, and so on. We consider an abstract to be structured if it has at least one such heading code.

3.4.6 *Subjective Additions:* :I, :U

The codes described so far aim at codifying repeatable properties, where several well-trained coders will come to the same result with high probability. In addition, we defined a number of suffixes for codes, by which coders can provide additional information for which the expectation of agreement is much lower. These are also not neutral, like the codes themselves, but all describe some kind of deficiency. The most important of these suffixes are the following:

Informativeness gaps (coded as :I) are spots in a sentence where the coder desired to know additional detail that is presumably available to the authors and that can presumably be provided in very little space. Example (from [LiuFenYin22]): *"To evaluate DeepState, we conduct an extensive empirical study on popular datasets and prevalent RNN models containing image and text processing tasks."* This sentence was coded as METHOD:I2, because the coder asked themselves *'How many datasets? How many RNN models?'* The answers to both are given in that article's Table 3: four datasets, three models. The authors could and should have given that information in the abstract.

Understandability gaps (coded as :U) are spots in a sentence where the coder encountered a term they did not know and found that the intuitive partial understanding they could muster for that term to be insufficient for understanding the abstract overall. Example (from [CheHuWei22]): *"Finally, we propose a new dynamic vocabulary strategy which can effectively resolve the UNK problems in code summaries."* This sentence was coded as DESIGN:U1 because it is unclear from the abstract what these apparently important *"UNK problems"* are supposed to be.

3.5 Training

The training phase (internally called "prestudy2") served two purposes: Finding/repairing deficiencies in the codebook, and arriving at a joint interpretation of it across the four coders (the four authors[4]). As can be seen in Figure 1, the training phase extended over almost half a year and triggered the majority of the codebook improvements.

In the training phase, we perfected the mechanics of the coding process (described in Section 3.7) and generally formed as a research team.

3.6 Sample

We decided not to aim for a broad selection of all software engineering research, but rather concentrate on what is presumably the highest quality material: The ICSE technical research track, the three journals allowed for journal-first presentations at ICSE, i.e., Empirical Software Engineering (*EMSE*), ACM Transactions on Software Engineering and Methodology (*TOSEM*), IEEE Transactions on Software Engineering (*TSE*). Since we expected to find that structured abstracts had better quality than unstructured ones, we added a fifth venue that *requires* structured abstracts: Information and Software Technology (*IST*, an Elsevier journal). IST has published many very good systematic literature reviews and methods works, but is not *generally* considered a top-quality venue.

We wanted to draw a random sample of 100 articles per venue from the 2022 volumes, but found that TOSEM had published only 86 articles that year, so we ended up at 486 articles initially. Of those, a few later had to be removed because they were not research articles (often editorials). Furthermore, we eventually did not need quite as much data

3. DESIGN has the most detailed definition of all our codes: 170 words.

4. Four additional people were involved in the training phase at some point, but they decided not to join the full study.

TABLE 1
Topics, codes, their descriptions (see our codebook for full descriptions), and how often we used them to code sentences.

Topic / Code	Short Description	Occurrences
Background **(b)**		
BACKGROUND	Context information	2003
H-BACKGROUND	Heading (e.g., *"Background:"* or *"Context:"*)	164
Gap **(g)**		
GAP	Unknown or not-yet-possible things, leading over to OBJECTIVE	538
NEED	Research that needs to be done (postulated), leading over to OBJECTIVE	26
Objective **(o)**		
OBJECTIVE	Top-level research goal, interest, or question	907
H-OBJECTIVE	Heading (e.g., *"Objective:"* or *"Aim:"*)	162
Design **(d)**		
DESIGN	Design, design process or features of an artifact (e.g., software, process, method)	1091
A-DESIGN	DESIGN announcement (e.g., *"A description of the tool is presented."*)	18
Method **(m)**		
METHOD	Approach or setup of an empirical (or possibly purely mathematical) study	1172
H-METHOD	Heading (e.g, *"Method:"*)	162
A-METHOD	METHOD announcement (e.g., *"A series of experiments is conducted."*)	33
Result **(r)**		
RESULT	Immediate, empirical outcome of the study	1494
H-RESULT	Heading (e.g, *"Results:"*)	160
A-RESULT	RESULT announcement (e.g., *"We identify key features."*)	105
CLAIM	Non-empirical would-be RESULT statement (e.g., *"This enables highly accurate code completion."*)	29
Summary **(s)**		
SUMMARY	Summarization of results, but no new information.	79
Conclusion **(c)**		
CONCLUSION	Take-home message, less specific than one or more RESULTs.	369
H-CONCLUSION	Heading (e.g, *"Conclusions:"*)	154
A-CONCLUSION	CONCLUSION announcement (e.g., *"We summarize recommendations."*)	42
Outlook **(o)**		
FPOSS	Research that is now possible.	83
FNEED	Future research that should be done.	50
A-FPOSS	FPOSS announcement (e.g., *"We propose future studies on the topic."*)	36
A-FNEED	FNEED announcement (e.g., *"Our findings emphasize the need for future research."*)	4
H-FWORK	Heading (e.g, *"Future work:"*)	2
:I	Informativeness gap (subjective assessment, see Section 3.4.6)	599
:U	Understandability gap (subjective assessment, see Section 3.4.6)	134
IGNOREDIFF	Inherent ambiguity (coders agree to disagree, see Section 3.7.2)	48

for answering our questions and stopped coding after 362 abstracts.[5] Still, ours is the largest manual study of abstracts we know of.

Volume downloading, sampling, and abstract extraction into publishable and annotation-ready text files were all done automatically by the scripts `retrievelit`[6], `select-sample`, and `prepare-sample` — except that the EMSE article format required manual cleansing. All these tools were purpose-built for the present study.

3.7 Coding Process

We coded each abstract twice, by so-called coders A and B. Abstracts are held in text files in separate directories `abstracts.A/` and `abstracts.B/`. Each sentence is followed by a line containing a pair of double curly braces `{{}}` into which the coder would enter their codings, such as `{{method,result:i2}}` (for a complex sentence that contains substantial amounts of method information as well as results with two informativeness gaps). We batched the coding in blocks of 8 abstracts each. Coders picked and processed blocks based on their available time, resulting in different numbers of blocks done by each author, between 13 blocks for Franz and 31 blocks for Lutz. The procedure, coordinated via git, is best explained by example, which we do in the following two subsections.

3.7.1 Coding

Step 1. When Lloyd wanted to code a block of abstracts on 2023-05-26, he found the next available block to be Block 17, which was already coded once by Lutz ("coder A"). Lloyd reserved his spot as "coder B" in the coordination file `sample-who-what.txt` and performed the coding.
Step 2. He then ran the `check-codings` script to test his codings against the codebook and corrected any mistakes, such as typos.
Step 3. He then ran `compare-codings` to compare his codings against Lutz'. This script creates one *report block* for each sentence where the codings of coders A and B are not compatible. We define codings to be compatible if they differ at most in the subjective suffixes :I and :U, but not in the codes (and also not by more than one in the numbers of informativeness gaps and understandability gaps). If Lloyd found a report block where Lutz' coding was obviously correct and his own obviously wrong, he would simply

5. After abstracts were dropped, our sample is no longer perfectly balanced, with EMSE:73, ICSE:74, IST:71, TOSEM:71, TSE:73 abstracts.
6. https://github.com/serqco/retrievelit/

correct his coding.
Step 4. He would then commit his coded abstracts into git.

Listing 1. Example of a coded abstract: [RosClaMad22].

```
Empirical Effort and Schedule Estimation Models for Agile
Processes in the US DoD.

Estimating the cost and schedule of agile software projects
 is critical at an early phase to establish baseline
budgets and schedules for the selection of competitive
bidders.
{{background}}
The challenge is that common agile sizing measures such as
story points and user stories are not practical for early
estimation as these are often reported after contract award
 in DoD.
{{gap}}
This study provides a set of effort and schedule estimation
 models for agile projects using a sizing measure that is
available before proposal evaluation based on data from 36
DoD agile projects.
{{objective,method}}
The results suggest that initial software requirements,
defined as the sum of functions and external interfaces, is
 an effective sizing measure for early estimation of effort
 and schedule of agile projects.
{{conclusion}}
The models' accuracy improves when application domain
groups and peak staff are added as inputs.
{{conclusion}}
---
```

3.7.2 Handling Disagreements

Step 5. For the remaining (unresolved) report blocks, Lloyd would write an email to Lutz explaining his reasoning.
Step 6. Lutz would read through that email and categorize the report blocks into the following cases:

a) Lloyd's coding is obviously correct (and Lutz' own is a clerical error, as in Figure 2) or Lutz prefers Lloyds coding. Lutz adjusts his coding accordingly.
b) Lutz finds Lloyd's coding clearly incorrect (clerical error) or Lutz prefers his own coding. He would respond with an explanation of his reasoning and suggest that Lloyd either adjust his coding or add an -IGNOREDIFF marker to it. This suffix indicates two accepted alternative interpretations of the same sentence.[7]
c) Lutz finds both codings equally acceptable. He would add an -IGNOREDIFF marker to his own and respond accordingly, explaining his reasoning.

Step 7. Lutz would commit his corrected abstracts and send the response email.
Step 8. Lloyd would read the response email and usually act on it to finish the handling of Block 17. Only rarely would he disagree with something to a degree that would make another round of emails necessary.

3.7.3 Effect of this Procedure

Most research involving content analysis codes most of their raw data only once, then codes a random subsample a second time, computes some coefficient of agreement, reports it as a measure of good-enough coding quality, and that's that. In contrast, our above-described procedure has two effects:

1) It maximizes the quality of the coding. Very few clerical errors, if any, will have managed to escape our discussion process. Besides, the definitions of all codes that are sometimes difficult to tell apart have been refined until they were very mature.
2) It finds all those spots in the sample where the abstracts are so convoluted or strangely formulated that even our careful code definitions do not lead to a canonical judgment. These spots, which will be marked by a -IGNOREDIFF annotation in our data, should clearly be considered to be badly written. In this manner, our coding produces additional findings of a relevant type.

Note that, technically speaking, our procedure also leads to a 100% inter-coder agreement, because even the cases of -IGNOREDIFF indicate that the coders agree on multiple plausible interpretations of one sentence.

3.8 Statistical Evaluation

The difficult part of our statistical evaluation is asking the right questions; our data provides a lot of possibilities. In contrast, the actual statistical techniques are simple and straightforward: Mostly tabulations of counts or percentages, bar plots, and box plots. We define two binary properties of abstracts: A *complete* abstract contains all of the basic elements described in Section 3.4.1. A *proper* abstract is complete and does not have any of the deficiencies described in Section 4 as making an abstract *improper*.

Since we coded each abstract twice, we get potentially conflicting assessments regarding these binary properties. We calculate and report percentages across all *codings* ($n_C = 724$, which each count as a "half" abstract), as well as lenient and conservative percentages per *abstract* ($n_A = 362$), for which either both or just one coder are necessary to attest a deficiency, respectively. We report these numbers with $\pm$ in the text and as error bars in the plots. When we write *'29%±0.6% of the abstracts are complete'*, we mean that 29% of the 724 abstract codings qualify the corresponding abstract as *complete*, and that for 28% of the 362 abstracts both coders agreed, while at least one coder thought so for 30% of them.

We mostly refrain from performing significance tests or computing confidence intervals, because most analyses are exploratory, not driven by specific expectations or theories. The one exception from this rule is the comparison of structured versus unstructured abstracts (Section 4.13).

Most phenomena we report are gradual by nature. In this spirit, we use the following verbal terms for frequencies: rare (less than 5%), not rare (5-20%), common (20-35%), frequent (35-50%), dominant (over 50%).

3.9 Interpretation

Our interpretation of the measurements is driven by our research interest and guided by our expertise as software engineering researchers who read abstracts. Whenever we mark a phenomenon as a weakness, we will provide a justification from that angle and provide an example to allow the reader to relate to the justification.

3.10 Readability Metric

For evaluating general readability at a purely language level, we use the Flesch-Kincaid "reading ease" readability score [23]. It is a validated and widely used metric that judges readability of English text based only on the number

7. It silences the `compare-codings` script for this particular report block, so that the coding difference is now officially accepted.

```
1  abstracts/abstracts.A/MeyAlmKel22.txt  (Lutz, Block 17)
2  abstracts/abstracts.B/MeyAlmKel22.txt  (Lloyd, Block 17)
3  [8] We found that (1) vulnerabilities related to improper resource control (e.g., session
4  fixation) are discovered faster and more often, as well as exploited faster, than vulnerabilities related to improper
   access control (e.g., weak password requirements), (2) there is a clear process followed by penetration
5  testers of discovery/collection to lateral movement/pre-attack.
6  {{method:i3}}  (Lutz)
7  {{result:i3}}  (Lloyd)
8
9  I think these are results. Perhaps your "method" code here is just a mistake.
10
11 Indeed a mistake.
```

Fig. 2. Excerpt from Lutz' response email during the disagreements handling for Block 17. Lines 1–7: Report block generated by the `compare-codings` script (**Step 3**); line 9: text line from Lloyds first email (**Step 5**); line 11: Lutz' response (**Step 6**). This one was a simple case; sometimes each coder provided several sentences of argumentation. Overall, Lloyd's first email contained report blocks for 9 disagreements in 4 abstracts, both typical numbers.

of words per sentence and the number of syllables per word. Values under 30 represent graduate-level difficulty, under 10 extreme difficulty for native speakers. Considering that most community members are not native speakers of English, we judge 20 to 30 to be a range suitable for researcher audiences and so we call anything over 30 "good", 20 to 30 "normal", and under 20 "improper".

3.11 Use of Examples

We will use examples from real abstracts from our sample to illustrate some of our statements. We identify their source by a citation key formed from three letters each of the first three authors' names. For example article 1 in Block 1 in our study was written by Fregnan, Petrulio, Di Geronimo, and Bacchelli and is thus identified as [FrePetGer22].

We select those examples for the clarity of the phenomenon in question, not for the abstract's quality. For the source of every positive example there are others that are as good or better. For the source of every negative example there are others that are as bad or worse. Since the use of an example is not about the article it stems from, we do not include those articles in our references list.

4 Results

4.1 Length and Readability

Typical abstracts (the middle half) are 190 to 280 words long and 8 to 14 sentences long. Structured abstracts tend to be longer than unstructured ones. Some venues have official abstract length limits: 150–250 words for EMSE, maximum 300 for IST, and recommended maximum 250 for TSE. These limits are disobeyed by more than a quarter of all articles for each of these three venues.

For readability, see Figure 3: only about 13% of all abstracts have good readability (>30), about 35% have normal readability (20-30), and 52% are improper (<20) — more than every fifth abstract (21%) even has a score below 10. ICSE tends to be better than the other venues, TOSEM is worse.

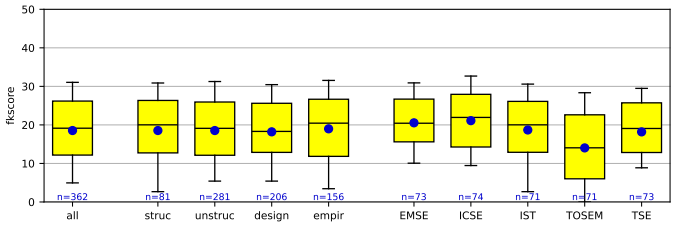

Fig. 3. Flesch-Kincaid 'reading ease' readability score, higher is better. Values under 50 are considered difficult to read (college-level material); under 30: very difficult to read (graduate level, acceptable for abstracts); under 10: extremely difficult to read (overly difficult). Differences between subgroups are modest. ICSE is best (mean 21), TOSEM is worst (mean 14). The abstract of the present article has a score of 33. Structured abstracts are just as difficult as nonstructured ones if one ignores the "Methods:" etc. headings as we did for the readability analysis.

4.2 Design Articles vs. Empirical Articles

We described the idea of the DESIGN code in Section 3.4.2. But many empirical works involve some artifact design as well, so how is the discrimination made? That depends on how the authors phrase their goal in their OBJECTIVE statement. Consider the following goal statements: *"In this paper, we propose AI-based automation for the completeness checking of privacy policies."* (from [AmaAbuTor22]). *"This study aims to investigate the urgency and importance of reproducibility and replicability for DL studies on SE tasks."* (from [LiuGaoXia22]).

The first puts the artifact at the center, so this is a design article. The second puts empirical results at the center, so this is an empirical article and the DESIGN code will *not* subsequently be used. Artifact design discussions are then usually coded as METHOD instead. In mixed cases, which are rare, the two coders would decide which aspect has more weight.

In design articles, a large fraction of the abstract will be devoted to describing design considerations for that artifact – typically 14% to 33% of the words in our data. The empirical study, which commonly exists as well, then usually has a mere supporting role (validating the claims made in the artifact discussion) and is correspondingly given less space: 5%–13% (versus 11%–24% for empirical works) for description of empirical method, 10%–22% (versus 16%–

33%) for description of empirical results. Design article abstracts are barely longer than empirical article abstracts.

4.3 The Abstracts Archetype

Compared to the content structure assumed by the usual formats of structured abstracts, our codebook is more fine-grained; the DESIGN code is but one example.

During our sensemaking process, we had a number of insights regarding how a well-written abstract "ticks", which we eventually distilled into the following template, which we call the *SERQco software engineering abstracts archetype*[8] (see also Figure 4):

1) An abstract consists of three parts, in this order: *Introduction*, *Study Description*, and *Outlook*.
2) Two turning points connect the three parts:
   a) A statement of the study goals (OBJECTIVE) connects *Introduction* to *Study Description*.
   b) A generalizing statement ("take-home message", CONCLUSION) connects *Study Description* to *Outlook*.
3) The *Introduction* first introduces the topic area of the study and what is known (BACKGROUND) and then may or may not point out a gap in knowledge (GAP).
4) For an empirical article, the *Study Description* begins with method description (METHOD), followed by results description (RESULT). Sometimes, this sequence occurs twice in a row, very rarely more.
5) For a design article, design description (DESIGN, see below) precedes the structure described in the previous item.
6) After the CONCLUSION, the *Outlook* talks about future research and states what could now be done (FPOSS, for future possibilities), what should now be done (FNEED), what the authors themselves intend to do (FWORK), or what is still not known (FGAP). Several statements of each type may occur (including none), in no particular order. In most cases, the space devoted to *Outlook* would be more informative if spent on *Study Description*.

The archetype describes all variants of abstracts that have a natural train of thought. It is an engineering-specific generalization of the well-known IMRAD structure [8]. Deviations from the archetype will tend to lead to a less easily understandable abstract.

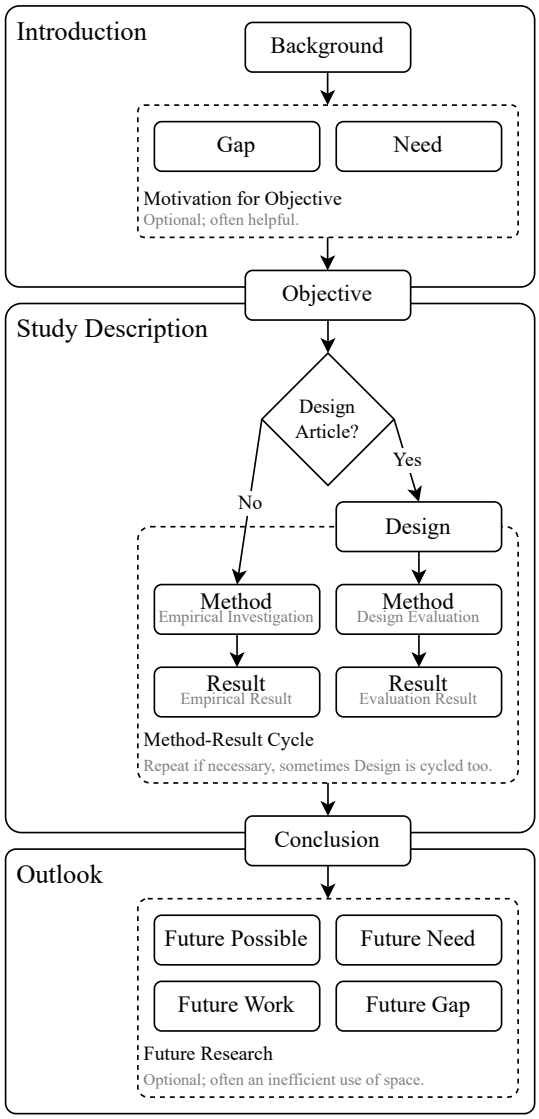

Fig. 4. A visual representation of the SERQco abstracts archetype. Note that this figure only describes the "natural train of thought" and not all possible abstract forms that we coded in this study. This is the recommended structure.

4.4 How Not To: Inefficient Allocation of Space

If one reads the abstracts in our sample, one can hardly help notice that some of them spend a lot of space on BACKGROUND, although its only purpose is to situate the objective and make it understandable. For example, [FirFirRos22] spends 66% of the abstract for explaining their rather niche application domain in some detail rather than just naming it and letting interested readers look up the rest in the article body.

As we can see in the Background group of boxplots in Figure 5, all venues that do not require structured abstracts have at least a quarter of articles that spend over 30% of the abstract space on background. As we can see in Figure 7, this leads to deficiencies later on: The conclusion is the potentially most useful part of the abstract, the take-home message, but with a long background section, it tends to become pronouncedly shorter.

Background lengths are more benign for structured abstracts.

4.5 How Not To: Missing Elements

Given the archetype, one way to approach an analysis of abstracts quality is to ask how often key parts of an abstract are missing entirely. This is shown in Figure 6.

The GAP and *Outlook* parts FPOSS, FNEED, etc. are clearly optional, so it is not a problem that their absence is frequent (40%) and dominant (79%), respectively. BACKGROUND and OBJECTIVE are rarely missing (2% and 3%).

Missing METHOD and RESULT are not rare (11% and 10%), which we find alarming. Both are always present in our structured abstracts.

The shocking part of this analysis is CONCLUSION, which ought to be present as the key take-home message in any

8. SERQco is the Software Engineering Research Quality Coalition

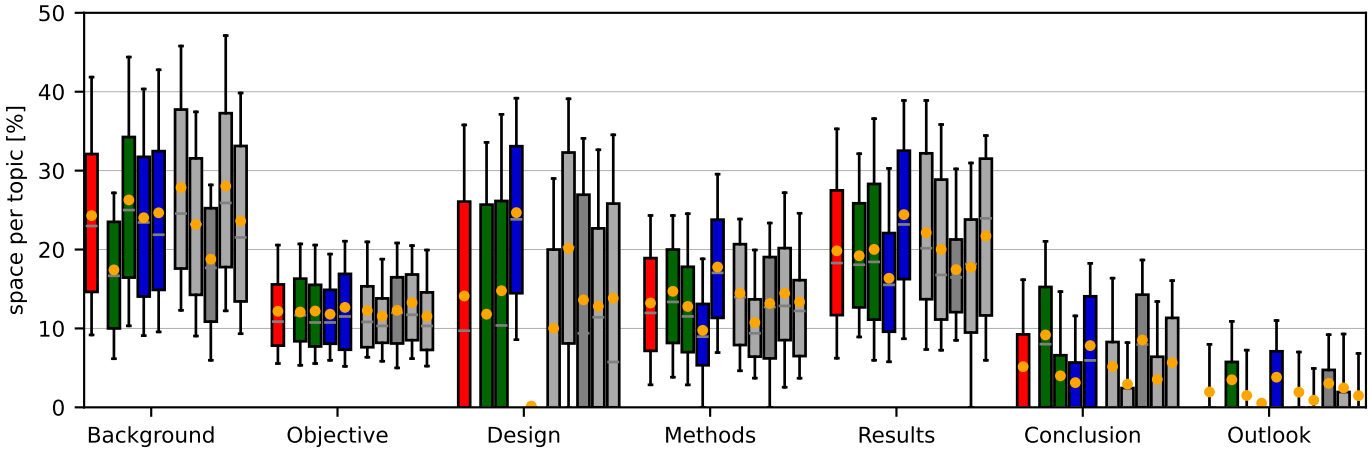

Fig. 5. Per-topic distribution of the amount of space used for that topic. These "topics" are groupings of related codes (see also Table 1); e.g., Outlook stands for the union of FPOSS, FNEED, and all corresponding A-* and H-* codes.
The box shows the 25-to-75 percentile, the whiskers are 10- and 90-percentile, the gray bar is the median, the fat dot is the mean.The plots in each group show these different subsets of abstracts from left to right: all (red); structured, non-structured (green); design, empirical (blue); EMSE, ICSE, IST, TOSEM, TSE (gray).

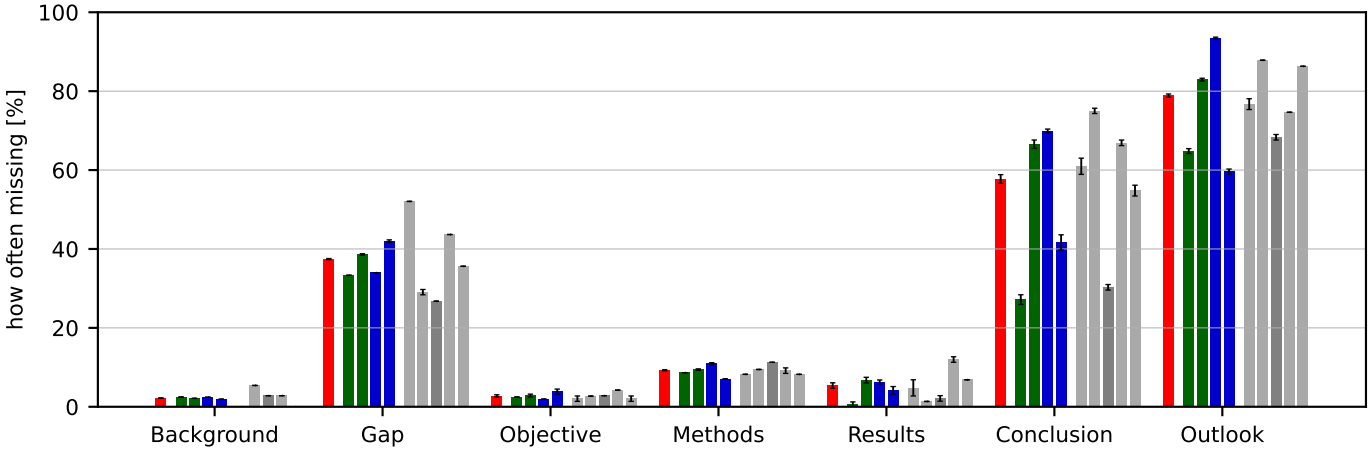

Fig. 6. How often is a topic not present at all in an abstract? These "topics" are groupings of related codes (see Table 1).
The plots in each group show these different subsets of abstracts from left to right: all (red); structured, non-structured (green); design, empirical (blue); EMSE, ICSE, IST, TOSEM, TSE (gray). Error bars are due to ambiguous formulations (see Section 3.7.2).

abstract, but is in fact missing in more than half of all (62%).[9] The situation is better for structured abstracts, but missing CONCLUSION are still common here (29%).[10]

Overall, only 29%±0.6% of all abstracts are *complete* in the sense that they contain all basic elements BACKGROUND (or GAP), OBJECTIVE, METHOD, RESULT, and CONCLUSION.

4.6 How Not To: Confusing abstracts

Any deviation from the natural train of thought of the abstracts archetype will tend to increase the reader's cognitive load. If this happens, we call the abstract's train of thought convoluted.

For investigating this issue, we map each abstract's structure to a sequence of letters, where each letter stands for a contiguous stretch of sentences in the abstract that have the same topic (see Table 1).[11] For instance, an abstract that follows a minimal incarnation of the archetype for an empirical article would be encoded as **`bomrc`**, which stands for the topic sequence *background, objective, methods, results, conclusion*. Non-minimal incarnations allow for many different structures but even complex structures (with long strings) *can* be easy to read. Other structures, however, can be problematic. The full list of abstracts structures is too long to discuss it here: It has 117 entries for empirical articles and 125 for design articles.

We have found no simple criterion to reliably diagnose which of these to consider convoluted and which not, so this section will only provide examples and does not quantify how many abstracts are actually convoluted. Furthermore, even abstracts that *do* conform to the archetype can be

9. This value is higher than the 'all' bar in Figure 6, because that bar shrinks when an H-CONCLUSION or A-CONCLUSION appears, but those do not serve the CONCLUSION purpose.

10. The prompt of having to write something after a *"Conclusion:"* keyword cannot guarantee an actual CONCLUSION: Dominantly, the statement authors put there is actually a RESULT (an immediate empirical result), a SUMMARY (a result repetition) or still something else.

11. However, if a, say, H-CONCLUSION is followed by, say, a SUMMARY (which is a common style), this would result in a topicletter string that looks more complicated than the abstract actually reads. We find this misleading and hence exclude H-* codes from the topicletter string.

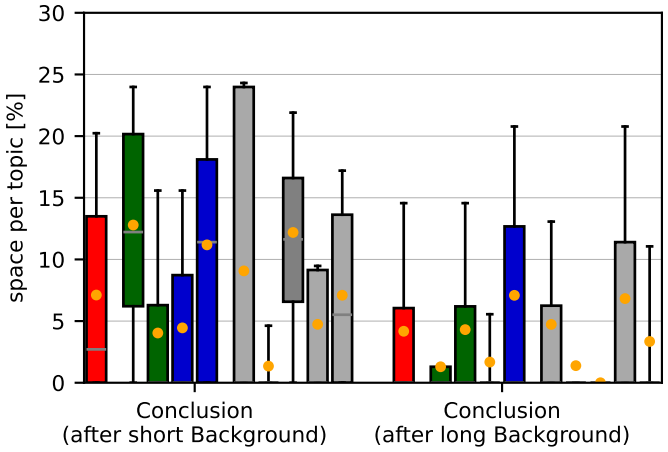

Fig. 7. Comparison of the amounts of remaining space devoted to the conclusion for abstracts with a rather short background section (lowest quarter) vs those with a long one (highest quarter). The latter conclusions are shorter even though the indicated percentage pertains to the part of the abstract after the background only.
The plots in each group show these different subsets of abstracts from left to right: all (red); structured, non-structured (green); design, empirical (blue); EMSE, ICSE, IST, TOSEM, TSE (gray).

confusing for semantic reasons. We will include examples of this type as well.

Abstract structures with more than two instances are shown in Figure 8 for empirical articles and Figure 9 for design articles. (Fractional frequencies such as 2.5 mean that the two coders did not agree.)

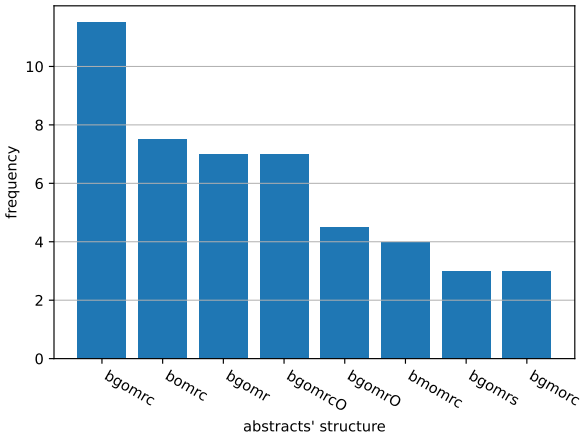

Fig. 8. The frequency of different trains-of-thought in the abstract for empirical articles. The label is a string of stretch-code characters: b-ackground, g-ap, o-bjective, d-esign, m-ethod, r-esult, s-ummary, c-onclusion, O-utlook.

4.6.1 *Structured Abstracts*

Structured abstracts help reduce confusion in two ways: They standardize the order in which ideas are presented, so readers know what to expect when. And their subheadings announce specifically what is to come next, which avoids many ambiguities.

Today's conventions for structured abstracts may be a bit restrictive (e.g. by not accommodating the useful **`mrmr`** substructure), but they do indeed result in more orderly

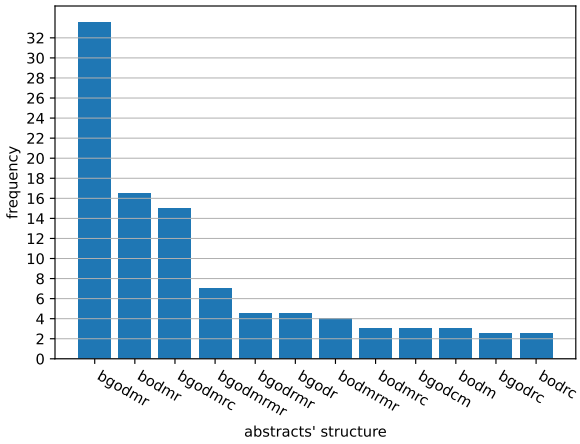

Fig. 9. The frequency of different trains-of-thought in the abstract for design articles. Same characters as before, except that d-esign can now in fact occur.

abstracts: There are only 31 different abstract structures for the 84 structured abstracts of empirical articles, but 56 different abstract structures on average for a random sample of the same size of the non-structured abstracts of empirical articles.

The orderliness of a structured abstract may not help, however, if the supposed structure is broken. We have seen only few cases of this for empirical articles, but once in a while some piece of information appears not where it should (such as BACKGROUND in the OBJECTIVE section in [LiaGaoXia22]) or authors appear to have added the subheadings to an already written non-structured abstract without rewriting it, so that lots of sections contain information of the wrong type (as in [SelOunSai22]).

The latter of these occurs in design articles as well (as in [GeFanQia22]). The former is even unavoidable for a design article, as no proper place for design information exists in today's conventional structured abstract section list.

4.6.2 *Empirical Articles*

As we see in Figure 8, there are only 8 among the 31 different abstract structures of empirical works that occur more than twice. Of these, bars 5, 6, 7 show abstracts with a non-archetypical train of thought: They mix METHOD and OBJECTIVE (**`bmomrc`** and **`bgmorc`**) or have a trailing RESULT after the CONCLUSION (**`bgomrcr`**).

When we annotated the abstracts, we kept notes on remarkable cases of all kinds. Combing through those notes, we can look for patterns of actually confusing abstracts. For the empirical works, there are some abstracts with information that is partially out-of-place or mis-shaped and will increase cognitive load for readers, but they are diverse and hardly worth calling a common pattern; for example: METHOD before OBJECTIVE ([PreMohVil22]), additional BACKGROUND late in the abstract ([UddGuéKho22]), a sudden second objective in the very last sentence ([DanPlaHer22]), an objective that sounds like from a design work ([GaoZhuZha22]), repeating (in different words) the objective where a conclusion should be ([AhmMerBah22]),

or mixing present tense and past tense such that results sound as if they were conclusions ([ValHunFig22]).

4.6.3 Design Articles

As we see in Figure 9, many of the common structures of design works lack some expected parts, most often a conclusion, which is missing for all bars except numbers 3, 8, and 10. Yet in terms of the *order* of things, only two structures deviate from the archetype: **bodm** (bar 9) and **bgodcm** (bar 10), both of which curiously end with METHOD information. Having multiple **mr** pairs is more common for design works than it is for empirical works.

As for truly confusing abstracts, the most conspicuous pattern is an unclear answer to the most fundamental question: Is this a design work or an empirical work? This issue occurs in various forms: a clear design OBJECTIVE, but a solely empirical CONCLUSION ([CorRweFra22]), the OBJECTIVE is phrased like for an empirical work, but the CONCLUSION makes clear it is a design work ([BocSchApe22], [SuFanChe22]), a clear design OBJECTIVE, but not a single DESIGN statement follows ([FirFirRos22], [HerFerGal22]), there are two OBJECTIVE statements, placed noncontinguously, and the first is clearly empirical ([MadNagBir22]), or many statements throughout have an unclear role ([NaeAla22]).

Of course, cases of misplaced information occur in design works as well, e.g. a RESULT in the middle of a very long BACKGROUND section ([NiuWuNie22]), BACKGROUND placed behind OBJECTIVE, which makes it sound a lot like a RESULT ([LuLiLiu22]), METHOD and RESULT placed after H-CONCLUSION ([HumKho22], [ImrDam22]).

Other peculiarities appear in design works that we have not seen in empirical works, e.g. an OBJECTIVE that is difficult to recognize as such ([TiaLiPia22]), formulating a CONCLUSION without ever presenting a RESULT ([LiaHanLi22]), or a sequence of OBJECTIVE, OBJECTIVE, DESIGN+RESULT phrased as a first... then... finally structure ([GueLarChe22]).

4.7 How Not To: Uninformative Formulations

Another way of reducing the usefulness of an abstract is using formulations that fail to provide information that, at this point, would be useful to the reader and could be provided using only very few additional words.

Our investigation has identified and then quantified two types of such lacks of informativeness. We call them informativeness gaps and announcements, respectively.

4.7.1 Informativeness Gaps

Look at the following result statement: *"random sampling is rare"* [BalRal22]. If at this point the reader expects the work contains something more concrete than *"rare"*, this is an informativeness gap. And indeed the article in question contains this information in its Table 4 and therefore could and should have said *'random sampling is rare **(8% of cases)**'*.

Informativeness gaps (coded :I, see Section 3.4.6) appear mostly in RESULT statements (83% of the gaps) or METHOD statements (12% of the gaps). Most (if not all) of them could be filled by a number. They tend to cluster, like in [ShiBiaBri22]: *"The results show that PRINS can process large logs much faster than a publicly available and well-known state-of-the-art tool, without significantly compromising the accuracy of inferred models."* This sentence has three informativeness gaps. A better formulation could have been: *'The results show that PRINS can often process logs **an order of magnitude** faster than the well-known state-of-the-art tool **MINT**, but **never lost more than 7 percentage points** of balanced accuracy.'*

More than half of all abstracts (55%±11%) have an informativeness gap (making them *improper*) and 18%±5% have three or more. See the leftmost two groups of Figure 10 for details.

4.7.2 Announcements

Occasionally, a sentence will not merely miss to report some specific piece of information but rather fail to provide any useful information at all and merely hint at information to be found in the article body. We call such a sentence an announcement (coded A-*, see Section 3.4.4).

Example (A-METHOD from [BesMarBos22]): *"As a second step, this study sets out to specifically provide a detailed assessment of additional and in-depth analysis of technical debt management strategies based on an encouraging mindset and attitude from both managers and technical roles to understand how, when and by whom such strategies are adopted in practice."* Here is what the sentence could have said instead: *'We then surveyed 26 managers and 46 technical people, followed by clarifying interviews with 4 managers and 2 developers in order to understand how the managers perceive how they are encouraging developers to manage technical debt and how the developers perceive the encouragement they receive.'*

Example (A-RESULT from [FliChaDye22]): *"Finally we provide guidelines/best practices for researchers utilizing time-based data from Git repositories."* This should have been a result statement such as the following: *'We provide 6 guidelines. For instance, cutting off all commits before 2014 will get rid of about 98% of all bad commits.'*

Announcements are a waste of space and a nuisance for the reader, yet 24% ±0.6% of all abstracts have at least one and we consider those *improper*. See groups three to six of Figure 10 for details.

4.8 How Not To: Undefined Important Terms

It is normal that the reader of an abstract has only a fuzzy understanding of what certain terms in the abstract mean. In a good abstract, the *approximate* meaning of a statement using such a term still comes across. Sometimes, however, this is not the case: The uncertainty regarding the meaning of the term weighs so much that the sentence containing it becomes incomprehensible. We call such a term use an understandability gap (coded :U, see Section 3.4.6) and consider such abstracts to be *improper*.

Here is an example from [HamMetQas22]: *"The results of evaluating the generality of the iContractML 2.0 reference model show that it is 91.7% lucid and 72.2% laconic."* Neither of the terms "lucid" or "laconic" have been introduced before, so that this sentence has two understandability gaps and an average reader is not able to make sense of this important statement which represents half of the work's results.

See the rightmost group of Figure 10 for how frequent this is: About 16%±10% of all abstracts have one or more

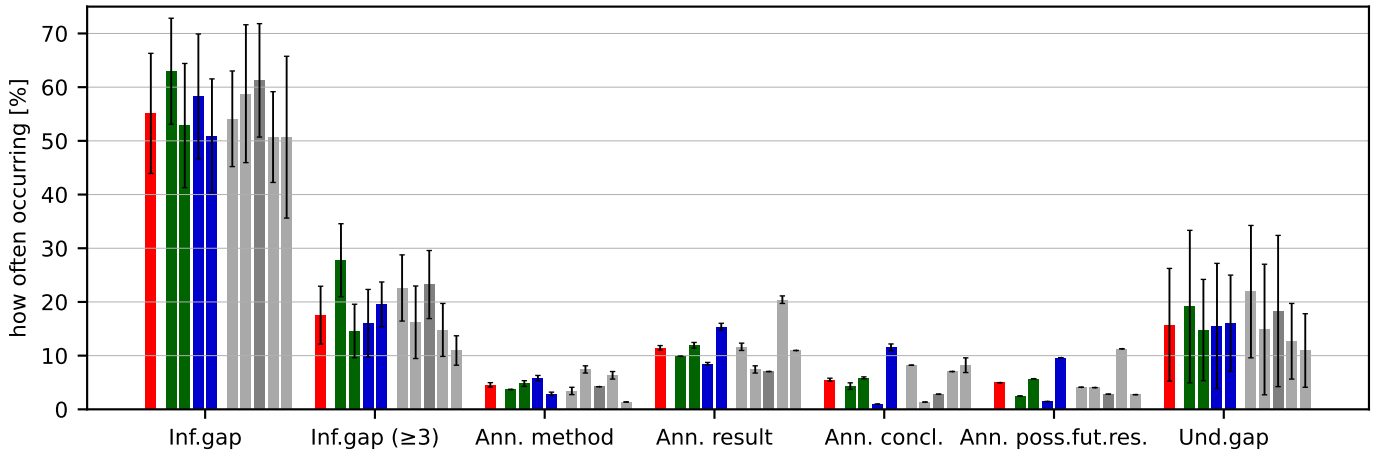

Fig. 10. What fraction of abstracts has the following uninformative types of formulations?
The plots in each group show these different subsets of abstracts from left to right: all (red); structured, non-structured (green); design, empirical (blue); EMSE, ICSE, IST, TOSEM, TSE (gray). The groups are: One-or-more or three-or-more informativeness gaps (i.e., missed opportunities for being more specific); some sentence that only announces (instead of describing) a method, result, conclusion, or possible future research; one or more understandability gaps. Error bars in the *Inf.gap* and *Und.gap* groups are due to differing coder perceptions of informativeness and understandability gaps (see Section 3.4.6); error bars in the *Ann.* groups are due to ambiguous formulations (see Section 3.7.2).
Informativeness gaps are epidemic, worse(!) for structured abstracts (1st green bar). Announcing is generally not rare, in particular for results, and tends to be less pronounced at IST (3rd gray bar). Not explaining key terms is not rare and worst at EMSE (1st gray bar).

understandability gaps. The issue is most pronounced at EMSE.

4.9 How Not To: Ambiguous Formulations

Codings with an -IGNOREDIFF marker indicate cases where the coders could not agree despite discussion (see Section 3.7.2): the respective sentence (or sentence part) is so highly ambiguous that more than one role for it is plausible. Obviously, such ambiguous formulations do not represent good abstract writing and we consider such abstracts to be *improper*.

In our data, we found 48 such cases overall, spread over 39 different abstracts, so that 11% of all abstracts have one or more such ambiguous sentences.

4.10 How Not To: Summary

Summing up, a proper abstract should be *complete*, i.e., it should provide at least a minimal amount of information of types BACKGROUND (OR GAP), OBJECTIVE, METHODS, RESULTS, and CONCLUSION — the canonical IMRAD structure.

However, as we see in Figure 11, a majority of abstracts fails this very basic quality criterion, a depressing result.

Our moderately stricter criterion of being *proper* involves being *complete*, having acceptable Flesch-Kincaid readability (>20), and having neither informativeness gaps nor understandability gaps nor highly ambiguous sentences. Given this list of criteria, improper abstracts will happen from time to time even for careful authors. Nevertheless, we believe that a majority of abstracts could and should be *proper* in this sense. Yet what we find is that only 4%±1.7% of them indeed are. We expected to find many problems in abstracts but still find this outcome astonishingly bad.

TABLE 2
For each coding-based quality criterion, there is a possibility for disagreement between the two codings: For a negative criterion (italics), the lower value denotes the number of abstracts for which *both* codings indicate a negative evaluation, the higher value denotes the number of abstracts for which *at least one* coding indicates a negative evaluation.

Criterion	Count		Ratio
Readability (Section 4.1)			
good ($fk > 30$)	47		13%
acceptable ($fk \in [20, 30]$)	126		35%
improper ($fk < 20$)	*189*		*52%*
Inefficient Allocation of Space (Section 4.4)			
background $\geq \frac{1}{3}$	82.5	±0.5	23% ±0.1%
Completeness (Section 4.5)			
yes	105	±2	29% ±0.6%
no	*257*	±2	*71% ±0.6%*
Informativeness Gap (Section 4.7.1)			
no	162.5	±40.5	45% ±11%
yes	*199.5*	*±40.5*	*55% ±11%*
Announcements (Section 4.7.2)			
no	274	±2	76% ±0.6%
yes	*88*	±2	*24% ±0.6%*
Understandability Gap (Section 4.8)			
no	305	±38	84% ±10%
yes	*57*	*±38*	*16% ±10%*
Ambiguous Formulations (Section 4.9)			
no	323		89%
yes	*39*		*11%*
Properness (Section 4.10)			
yes	13	±6	3.6% ±1.7%
no	*349*	*±6*	*96% ±1.7%*

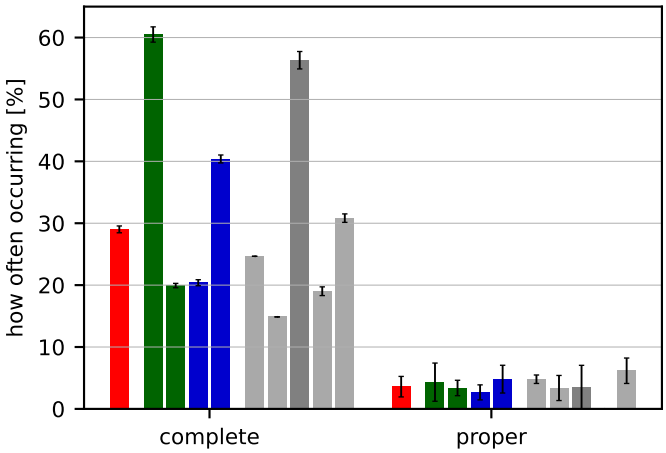

Fig. 11. What fraction of abstracts is *complete* in the sense of Sections 3.8 and 4.10? What fraction of abstracts is *proper* in the sense of Section 3.8?
The plots in each group show these different subsets of abstracts from left to right: all (red); structured, non-structured (green); design, empirical (blue); EMSE, ICSE, IST, TOSEM, TSE (gray). Error bars for *completeness* are due to ambiguous formulations (see Section 3.7.2), error bars for *properness* are due to differing coder perceptions of informativeness and understandability gaps (see Section 3.4.6).
Only 29%±0.6% of abstracts are *complete*. Only about 20% of unstructured abstracts and 21% of design article abstracts are complete. Structured abstracts are much better and therefore all four supposedly top-quality venues are beat by the much lower-regarded IST (3rd gray bar) which requires structured abstracts.
Only 4%±1.7% of abstracts are *proper*. Here, too, structured abstracts are better than unstructured ones. Our TOSEM sample has not a single proper abstract.

4.11 How To: Proper Abstracts

Indeed, the proper abstracts are so few, we can list them here completely.

- Empirical works with structured abstract: [AmnPoe22], [LavMor22], [TanFeiAvg22], [YuKeuXia22].
- Empirical works with non-structured abstract: [AbdBadCos22], [CinCooPas22], [HeMenChe22], [LinWilHal22], [OliAssGar22], [UddAlaSer22], [WanZhaZha22].
- Design works with structured abstract: [BaiJiaCap22], [WuSheChe22].
- Design works with non-structured abstract: [BarDuDav22], [CorRweFra22], [GalEwaJun22], [GerMarLat22], [HanMeh22], [YeGuMar22].

4.12 How To: Information-Rich Formulations

When avoiding the uninformative formulations of Section 4.7, some authors manage to pack large amounts of relevant information into a sentence. This most often occurs for METHOD or RESULT information.

Consider this METHOD sentence: *"Therefore, through a case study of 9 open source software projects across 30 versions, we study the relative effectiveness of SNA metrics when compared to code metrics across 3 commonly used SDP contexts (Within-project, Cross-version and Cross-project) and scenarios (Defect-count, Defect-classification (classifying if a module is defective) and Effort-aware (ranking the defective modules w.r.t to the involved effort))."* ([GonRajHas22])

This sentence is complex, but pays back for the reading effort by being *enormously* informative.

Or consider this well-designed METHOD+RESULT sentence: *"We evaluate the three versions on a set of 299 data constraints from 15 real-world Java systems, and find that they improve method-level link recovery by 30%, 70%, and 163%, in terms of true positives within the first 10 results, compared to their text-retrieval-based baseline."* ([FloPerWei22])

Easily readable, related information is kept together, and the meaning of results is explained precisely. We wished more authors would write like this!

4.13 How To: Structured Abstracts are more orderly

This study is mostly exploratory. Our only clear expectation at the start was that structured abstracts would tend to be better (whatever that was going to mean) than unstructured ones. Therefore, we perform two statistical hypothesis tests that compare structured to unstructured abstracts.

We find that structured abstracts are significantly more often *complete* ($\chi^2 = 49.9, p < 0.001$). For *properness*, the absolute numbers are so low that statistical significance is not achieved ($\chi^2 = 0.5, p = 0.48$) despite the trend visible in the relative difference of the two green bars in the right half of Figure 11. Nevertheless, overall the expectation appears to be correct.

5 Limitations and Threats to Validity

5.1 Interpretivist Perspective

In terms of Tracy's quality criteria for qualitative research [24], our study has nice properties, because our readers inhabit the domain of abstract reading and abstract writing themselves.

The topic's worthiness is obvious, the amount of data is large, as is the number of constructs used. Challenges are discussed below. Hopefully, our constructs and findings resonate with you; they certainly resonate with us. Should you find our description insufficiently thick, you can easily look up lots of additional examples in our raw data described in Section 3.2.

5.2 Positivist Perspective

Internal validity is the degree to which the stated method was followed correctly. We do not expect much problem in this regard. The most difficult-to-avoid problem in our study is lapses of concentration during coding which result in coding mistakes. However, the laborious coding procedure described in Section 3.7 makes it very unlikely for such mistakes to slip through. Most other steps were automated, so mistakes would be systematic and not likely to escape our attention.

Construct validity is the degree to which the design of the study is adequate for the phenomenon to be understood. Here, our study has obvious limitations: It ideally ought to measure the informativeness and understandability of abstracts. However, both of these are reader-dependent, so measuring them would involve a reading study with many readers. This would face huge problems in getting a representative set of readers, could never scale to the hundreds of abstracts we look at here, and, whichever operationalization it chose, it would be imperfect and controversial. We therefore decided to analyze properties of abstracts that

are *arguably* problematic, although we cannot say just how problematic in each case, resulting in count statistics only.

External validity is the degree to which the results generalize to other sets of abstracts: Here, we expect that our results generalize well to neighboring (past and future) years in the same venues we studied, perhaps a bit less for ICSE because of its varying location and hence more varying authors. Whether it also generalizes to other venues we cannot know, but we would be surprised if venues of lower scientific reputation had articles with much better abstracts.

6 Conclusions

6.1 Too Few Abstracts Have Good Quality

Only 29% of the investigated software engineering research article abstracts are *complete* according to the IMRAD structure long established for abstracts in science (Figure 11). In particular, more than half of all abstracts (62%) never formulate a conclusion in the sense of a generalizing take-home message (Figure 6). Only 4% of the abstracts are *complete* and also fulfill modest additional criteria of informativeness and understandability (Figures 3 and 10, Section 4.9). We define and quantify further quality issues not included in the above number (Figures 5, 7, 10, and non-quantitative Section 4.6). This low abstracts quality exists despite the fact that we analyzed only venues supposed to have high quality.

This is deplorable, because for the vast majority of readers, the abstract is the first part they read [2] — and often the only part. Futhermore, scientific indexes and specialized search engines such as *Scopus AI*[12] rely heavily on abstracts. With the above level of abstracts quality, readers will get far less well informed than they could have been and the spread of knowledge will be slowed down accordingly, needlessly wasting public money.

We conclude that the software engineering research community should pay more attention to abstract-writing. Presumably, introducing a structured abstract format and accompanying writing instructions that suit engineering research would help.

6.2 How To: Guidelines for Well-Written Abstracts

6.2.1 For Authors

The steps cross-reference the article sections that supply detail or evidence.

1) Write a structured abstract, not a free-flowing one (see Section 4.13) and follow the archetype (Sections 4.3). If the venue requires structured abstracts in an unsuitable format, protest.
2) Take care to avoid announcements (see Section 4.7.2), understandability gaps (see Section 4.8), and sentences with an unclear role (see Section 4.9). Provide helpful detail, perhaps simplified, if it requires only little space (see Section 4.7.1). Keep your sentences short.
3) Write a short Background section that provides just enough context and motivation to understand the subsequent Objective (see Section 4.4).
4) Decide on your main contribution. Is your article a design article or an empirical one? Write an Objective that expresses the type and your specific goal succinctly (see Section 4.6). If your background information contains a corresponding Gap statement, clearly phrase it as such.
5) If your work is a design article, write a Design section. Cover all key ideas (typically two to four), avoid non-key information.
6) If you have multiple near-independent empirical substudies and your work is a design article, use two or three combined Method and Results sections. Each of these will typically be a single sentence of the form "We do-this-and-that and find this-and-that." (see Section 4.6). Otherwise write separate Methods and Results sections as follows.
7) Write a Methods section that explains what you have done for your empirical study. Be as specific as you can do concisely. In particular, mention the amount of data used (see Section 4.7.1).
8) Write a Results section that explains the main outcomes of your study. If you have many results, report the one or two most important ones. If you have many results of equal importance, report the one or two most interesting ones or just provide examples. Be specific and beware of announcements (see Section 4.7.2).
9) Write a Conclusion section that generalizes from your results. What should the reader take home? What do we now know that we did not know before? The broader the conclusion, the higher the relevance of your work, but the lower its credibility. Find a formulation with good relevance and good-enough credibility. Do not be a coward (see Section 4.5): Dare to later be proven wrong for a few of your works.
10) An outlook on future possibilities *can* be part of your Conclusion, but is usually better left to the body of your article.

6.2.2 For Venues

Structured abstracts are not currently used widely in software engineering, presumably because their usual form does not suit design articles.

In a suitably extended form, however, structured abstracts promise better abstracts quality than a free-style format because our results strongly suggest that a structured format is helpful for completeness. Previous results show that it is also helpful for understandability. Therefore, venues should require structured abstracts in such a new, engineering-ready format as described by Table 1 and Sections 4.3 and 6.2.1: allowing Gap statements, allowing Design sections, and allowing multiple Methods and Results sections.

For your call for papers, feel free to copy the above text, point your readers to the online version,[13] perhaps point authors to this article for the underlying evidence.

Acknowledgments

We thank Gesine Milde for cleansing the automatically extracted abstract texts.

12. https://www.elsevier.com/products/scopus/scopus-ai

13. https://github.com/serqco/qabstracts/blob/main/serqco-abstracts-structure.md

REFERENCES

[1] T. A. Lang, "Scientific abstracts: Texts, contexts, and subtexts," *European Science Editing*, vol. 48, 2022.

[2] F. Shiely, K. Gallagher, and S. R. Millar, "How, and why, science and health researchers read scientific (IMRAD) papers," *PLOS One*, vol. 19, no. 1, 2024.

[3] A. Strauss and J. Corbin, *Basics of qualitative research*. Sage Publications, 1990.

[4] K. Krippendorff, *Content analysis: An introduction to its methodology*, 2nd ed. Sage Publications, 2004.

[5] J. M. Swales, *Genre analysis: English in academic and research settings*. Cambridge University Press, 1990.

[6] M. B. Dos Santos, "The textual organization of research paper abstracts in applied linguistics," *Text & Talk*, vol. 16, no. 4, pp. 481–500, 1996.

[7] C. Cross and C. Oppenheim, "A genre analysis of scientific abstracts," *Journal of Documentation*, vol. 62, no. 4, pp. 428–446, 2006.

[8] L. B. Sollaci and M. G. Pereira, "The introduction, methods, results, and discussion (IMRAD) structure: a fifty-year survey," *Journal of the Medical Library Association*, vol. 92, no. 3, p. 364, 2004.

[9] A. Dupuy, K. Khosrotehrani, C. Lebbé, M. Rybojad, and P. Morel, "Quality of abstracts in 3 clinical dermatology journals," *Archives of Dermatology*, vol. 139, no. 5, pp. 589–593, 2003.

[10] S. Sharma and J. E. Harrison, "Structured abstracts: do they improve the quality of information in abstracts?" *American Journal of Orthodontics and Dentofacial Orthopedics*, vol. 130, no. 4, pp. 523–530, 2006.

[11] A. B. Rosen, D. Greenberg, P. W. Stone, N. V. Olchanski, and P. J. Neumann, "Quality of abstracts of papers reporting original cost-effectiveness analyses," *Medical Decision Making*, vol. 25, no. 4, pp. 424–428, 2005.

[12] C. M. Faggion, Jr, J. Liu, F. Huda, and M. Atieh, "Assessment of the quality of reporting in abstracts of systematic reviews with meta-analyses in periodontology and implant dentistry," *Journal of Periodontal Research*, vol. 49, no. 2, pp. 137–142, 2014.

[13] W. Arthur, Z. Zaaza, J. X. Checketts, A. L. Johnson, K. Middlemist, C. Basener, S. Jellison, C. Wayant, and M. Vassar, "Analyzing spin in abstracts of orthopaedic randomized controlled trials with statistically insignificant primary endpoints," *Arthroscopy: The Journal of Arthroscopic & Related Surgery*, vol. 36, no. 5, pp. 1443–1450, 2020.

[14] V. Reynolds-Vaughn, J. Riddle, J. Brown, M. Schiesel, C. Wayant, and M. Vassar, "Evaluation of spin in the abstracts of emergency medicine randomized controlled trials," *Annals of Emergency Medicine*, vol. 75, no. 3, pp. 423–431, 2020.

[15] S. Jellison, W. Roberts, A. Bowers, T. Combs, J. Beaman, C. Wayant, and M. Vassar, "Evaluation of spin in abstracts of papers in psychiatry and psychology journals," *BMJ Evidence-Based Medicine*, vol. 25, no. 5, pp. 178–181, 2020.

[16] I. Boutron, S. Dutton, P. Ravaud, and D. G. Altman, "Reporting and interpretation of randomized controlled trials with statistically nonsignificant results for primary outcomes," *JAMA*, vol. 303, no. 20, pp. 2058–2064, 2010.

[17] D. Moher, S. Hopewell, K. F. Schulz, V. Montori, P. C. Gøtzsche, P. J. Devereaux, D. Elbourne, M. Egger, and D. G. Altman, "CONSORT 2010 explanation and elaboration: updated guidelines for reporting parallel group randomised trials," *International Journal of Surgery*, vol. 10, no. 1, pp. 28–55, 2012.

[18] J. Hartley, M. Sydes, and A. Blurton, "Obtaining information accurately and quickly: are structured abstracts more efficient?" *Journal of Information Science*, vol. 22, no. 5, pp. 349–356, 1996.

[19] B. A. Kitchenham, T. Dybå, and M. Jørgensen, "Evidence-based software engineering," in *Proc. 26th Int'l. Conf. on Software Engineering*. IEEE, 2004, pp. 273–281.

[20] B. A. Kitchenham, O. P. Brereton, S. Owen, J. Butcher, and C. Jefferies, "Length and readability of structured software engineering abstracts," *IET Software*, vol. 2, no. 1, pp. 37–45, 2008.

[21] D. Budgen, B. A. Kitchenham, S. M. Charters, M. Turner, P. Brereton, and S. G. Linkman, "Presenting software engineering results using structured abstracts: a randomised experiment," *Empirical Software Engineering*, vol. 13, pp. 435–468, 2008.

[22] L. Prechelt, L. Montgomery, J. Frattini, and F. Zieris, "How (not) to write a software engineering abstract (data package)," zenodo.org, DOI:10.5281/zenodo.15736253, https://doi.org/10.5281/zenodo.15736252, 2025.

[23] J. P. Kincaid, R. P. Fishburne, Jr, R. L. Rogers, and B. S. Chissom, "Derivation of new readability formulas (Automated Readability Index, Fog Count and Flesch Reading Ease Formula) for navy enlisted personnel," United States. Naval Education and Training Support Command. Chief of Naval Technical Training, Research Branch Report 8-75, 1975. [Online]. Available: https://stars.library.ucf.edu/istlibrary/56/

[24] S. J. Tracy, "Qualitative quality: Eight "big-tent" criteria for excellent qualitative research," *Qualitative Inquiry*, vol. 16, no. 10, pp. 837–851, 2010.

Lutz Prechelt received a PhD from the University of Karlsruhe for work that combined machine learning and compiler construction for parallel machines. He then moved to empirical software engineering and performed a number of controlled experiments before spending three years in industry as an engineering manager and CTO. He is now full professor for software engineering at Freie Universität Berlin and executive director of the Institute for Informatics there. His research interests concern the human factor in the software development process, asking mostly exploratory research questions and addressing them with qualitative methods. Additional research interests concern research methods and the health of the research system. He is the founder of the Software Engineering Research Quality Coalition (SERQco) and the inventor of Review Quality Collector (RQC). Contact him at prechelt@inf.fu-berlin.de.

Lloyd Montgomery received his PhD from the University of Hamburg, Germany, working under Prof. Dr. Walid Maalej. Currently, his primary research area is the quality of issue tracking systems, with other interests such as empirical software engineering, non-technical factors in software engineering, and science communication. He won the RE'17 best paper award for his machine learning design science research with IBM customer support. Lloyd's academic service record includes serving as the IST Publicity Chair, RE'23 Publicity Chair, NLP4RE'22 Workshop Co-Chair, and RE'21 Artefact Co-Chair. He is on the program committees of REFSQ, NLP4RE, and the RE@Next! track at RE, and regularly reviews for the journals REJ, IST, and JSS, in addition to reviews for SQ Journal and TOSEM. Lloyd also has a particular passion for artefact tracks, having served as a PC member for seven artefact tracks at RE, ICSE, and FSE. Contact him at lloyd.montgomery@uni-hamburg.de.

Julian Frattini (julian.frattini@chalmers.se) obtained his Ph.D. degree from Blekinge Institute of Technology (BTH), Sweden, and is currently a postdoctoral researcher at Chalmers University of Technology and University of Gothenburg, Sweden. He contributes research to requirements engineering with his work on requirements artifact quality, as well as to the field of software engineering research methodology with a particular interest in statistical causal inference and Bayesian data analysis. Julian served on the organizing committees of REFSQ, RE, AIRE, and CrowdRE, and is the publicity co-chair of the IST journal. He is on the program committees of RE, REFSQ, EASE, ESEM, QUATIC, and NLP4RE, and reviews for the journals TSE, TOSEM, EMSE, REJ, IST, and JSS.

Franz Zieris received a PhD from Freie Universität Berlin for qualitative research on pair programming in industy. He then spent 2.5 years in industry as software architect and business analyst. He is now an Associate Senior Lecturer at the Department of Software Engineering at BTH, where his research focuses on continuous software engineering and distributed software engineering. Franz is on the program committes of the Internal Conference on Agile Software Development (XP) and International Conference on Cooperative and Human Aspects (CHASE), and regularly reviews for Empirical Software Engineering (EMSE), the Journal of Systems and Software (JSS), and Information and Software Technology (IST).