How (Not) To Write a Software Engineering Abstract

Lutz Prechelt , Lloyd Montgomery , Julian Frattini, Franz Zieris

Abstract—*Background:* Abstracts are a very valuable element in a software engineering research article, but not all abstracts are as informative as they could be. *Objective:* Characterize the structure of abstracts in high-quality venues, observe and quantify deficiencies, suggest guidelines for writing informative abstracts. *Methods:* Use open coding to derive concepts that explain relevant properties of abstracts and identify the archetypical structure of abstracts. Use quantitative content analysis to objectively characterize abstract structure over a sample of 362!!! abstracts from five presumably high-quality venues. Use exploratory data analysis to find recurring issues in abstracts. Compare the prototypical structure to actual structures to derive guidelines for producing informative abstracts. *Results:* ((1: summarize results)) *Conclusions:* (1) Even in top venues, many abstracts are far from ideal. (2) Structured abstracts tend to be better than unstructured ones, but (3) artifact-centric works need a different structured format. (4) The community should start requiring conclusions that generalize, which currently are often missing.

Index Terms—!!!.

1 Introduction

Although the abstract is a super important part of any research article [1], 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.

The present article aims at substantiating this impression.

1.1 Research Questions

We ask several questions:

1) What does a typical well-written abstract look like?
2) Which deficiencies occur?
3) How often?
4) Do structured abstracts have better quality than unstructured ones?
5) How should software engineering abstracts be written?

Of these, numbers 1 and 2 should be considered exploratory, number 4 is hypothesis-driven (we expect a yes), and the answer to number 5 we consider a conclusion from the answers to the other four.

L. Prechelt is with Freie Universität Berlin, Germany
L. Montgomery is with Universität Hamburg, Germany
J. Frattini and Franz Zieris are with BTH, Karlskrona, Sweden

1.2 Research Approach

No firm expectations regarding questions 1 and 2 exist for engineering articles, so qualitative methods will have to be used for them: We start our research by open coding [2, Ch. 5] in order to derive a vocabulary (a set of concepts or codes) by which the nature of a particular abstract can be characterized.

We intend to convince even people who are skeptical of qualitative research regarding our answers to questions 3, 4, and 5 and regarding the need to improve the quality of software engineering abstracts. Therefore, we continue the research by performing a repeatable quantitative content analysis [3, Ch. 7] for number 3 on a large sample of 362!!! presumably high-quality abstracts and apply an elaborate seven-step approach for maximizing its reliability.

The final stage is the statistical evaluation of the content analysis data, which is again exploratory and is straightforward in both the questions it asks and the statistical methods it uses for answering them.

1.3 Research Contributions

Our contributions correspond to the research questions as follows:

1) As for the structure of well-written abstracts, we present an “abstract archetype” that describes fixed parts of the structure and degrees of freedom.
2) We describe and discuss !!! types of deficiencies.
3) We quantify the frequency of each deficiency type for the entire sample of abstracts as well as for 9 different subgroups of interest.
4) We present convincing data that structured abstracts tend to be better in several respects.
5) We provide data-based how-to instructions for abstracts writing for authors as well as guidance for editors and conference organizers. Software Engineering works should use structured abstracts, but need a different and more flexible template than what is used so far.

2 Related Work

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

2.1 Abstracts Structure

Swales introduced "genre analysis" as a means for teaching academic reading and writing especially to non-native speakers [4]: Genres are text types; 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 the writers for making their overall point. Several studies have looked at the move structure of research abstracts in different fields such as applied linguistics [5] or protozoology [6]. Despite the differences of research content, they find very similar moves, such as this five-move structure [6]:

> (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 suggests 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 [7].

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.

2.2 Abstracts Quality

Several studies on abstracts focus on quality assessment, most often in subfields of the biomedical domain. Many such studies cover articles of a homogeneous nature: all randomized controlled trials (controlled experiments). This allows formulating very specific expectations what information should be presented in an abstract and allows performing the analysis in checklist fashion, for example: In clinical dermatology, [8] used a 30-item checklist on 197 abstracts for computing a 0-to-1 score and found mean scores between 0.64 and 0.78 for their various subgroups. In dental medicine, [9] 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 [10]. Among 146 abstracts of meta-analyses in peridontology, 33% did not even report the direction in which the evidence was pointing [11].

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 ([12]), 44% in emergency medicine ([13]), 56% in psychiatry and psychology ([14]), and 58% for the conclusions alone across a broad set of medical controlled experiments ([15]).

Unfortunately, the methods of those studies are not applicable to a broad sample of software engineering works, 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 and 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 [6] 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 [8] nor [9] reports which of their checklist items are missing most frequently. Rather, their research question is the relative quality of structured versus unstructured abstracts. And in almost all of those cases, including [8] and [9], the answer is: structured abstracts have fewer quality issues.

A structured abstract is one that uses a prescribed sequence of intermediate headings, such as Background, Objective, Methods, Results, Conclusions or some similar set.

Such research can be highly influential. For instance, the CONSORTS report [16] (containing guidelines for reporting controlled experiments, with over 10,000 citations), relies on such a study [17] to recommend structured abstracts.

In software engineering, Kitchenham proposed Evidence Based Software Engineering (EBSE) in 2004 [18]. 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 study 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 [19].

The second, by Budgen, Kitchenham, and others [20], 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, which have very different (and more complicated) properties as we will see.

3 Methods

3.1 Overview

Our study is a full-blown content analysis in the sense of Krippendorff [3]: 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 started first and is described in Sections 3.3 and 3.4, defined the rules and target concepts of the counting. Training was for developing a joint understanding of the codebook and also contributed greatly to the codebook's early evolution; it is described in Section 3.5 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. Coding is described in Section 3.7. !!!Statistical evaluation

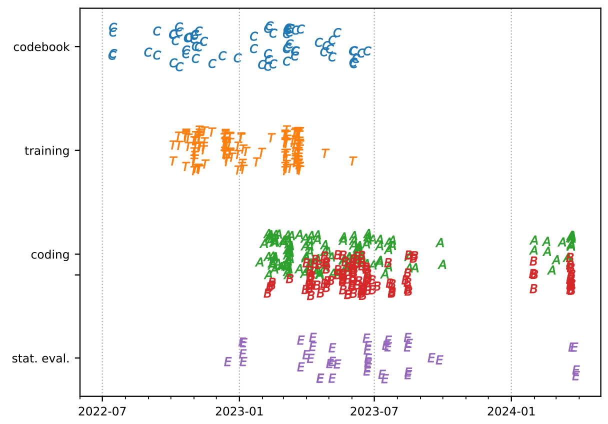


Fig. 1. Timeline of the main study phases and their individual events. Each character's x-coordinate represents the time of a git commit. The vertical scattering is added for legibility only.

Overall, we consider our study to be a qualitative one, but note that the middle two elements, coding and statistical evaluation, are compatible with a positivist epistemology (aiming for "objective" results) so 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. Find it at https://github.com/serqco/qabstracts/.

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: we code by sentence; we prefer single codes per sentence over multi-codings; for choosing a code, we act like readers and consider only what we have seen before plus one sentence forward context when needed and only when needed; be friendly and 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 is defined by a short verbal explanation. For example the definition of METHOD reads *"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. For example for METHOD, the part *"(or possibly purely mathematical)"* was initially not present and was added when we encountered the first of those (very few) purely mathematical studies during the coding process and were confused which code was appropriate for some sentence. To help disambiguation, a few of the definitions need to be much longer than the above.

3.4.2 *Artifact-centric Studies:* DESIGN

In a phase internally called "prestudy", the codebook was then refined and extended 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 predominantly talk about an empirical study. 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 contain an empirical study, are quite different in nature from purely empirical 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 works", the others "empirical works".

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

At various later points during the work, 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 sentences state what is unknown or not yet possible; SUMMARY sentences summarize several results, but

1. In Krippendorff's terminology, this is an "established theories" justification of analytical constructs [3, Section 9.2.3].

2. In Krippendorff's terminology, this is an "expert knowledge and experience" justification of analytical constructs [3, 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.

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

do not provide new information. In contrast to a CONCLUSION, a summary statement does not generalize beyond the immediate results. FPOSS, FNEED, FWORK sentences occur at the end of abstracts. They state what future work is now possible, needed, or planned-by-the-authors.

((2: examples of gap, summary, fposs, conclusion?))

3.4.4 *Codes for Announcements:* A-*

Sometimes, statements insinuate there will be certain information in the article body, but do not provide any concrete information themselves. We call such statements announcements and provide extra codes for them.

For example, here is an A-METHOD (method announcement) from [BosMarBos22]: *"As a second step, this study sets out to specifica[illegible] 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 also have codes for the headings (only the headings words themselves) 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 recognize structured abstracts by the presence of such a heading code.

3.4.6 *Subjective Additions:* :I, :U, :HYPE

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 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 are both 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 are spots in a sentence where the coder finds their usual intuitive half-understanding of a term used in the abstract insufficient for understanding the abstract overall.

((3: easy-to-understand example of :u1))

The :HYPE suffix marks statements that praise the article far more than warranted, often by using exaggerated adverbs such as "extremely". We have not applied this code consistently, so its uses only provide examples, but not measurements.

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 you can see 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 below, in particular the very useful `compare-codings` script and email routine, 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 (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 required 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 has published only 86 articles per year, so we ended up at 486 articles initially. A few of those later had to be removed because they were other things, often editorials. Furthermore, we eventually did not need quite as much data 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 retrievelit[6], select-sample, and prepare-sample scripts, respectively, except that the EMSE article format required manual abstracts cleansing.

3.7 Coding Process

We code each abstract twice, by so-called coders A and B. Abstracts are held in text files in separate directories

4. Four more people were involved in the training phase at some point but did not eventually join the study.

5. After abstracts were dropped, our sample is no longer perfectly balanced, but has EMSE:73!!!, ICSE:74, IST:71, TOSEM:71, TSE:73 abstracts. ((4: do we need to balance it?))

6. https://github.com/serqco/retrievelit/

`abstracts.A` and `abstracts.B`. Each sentence is followed by a line containing `{{}}` and into this pair of double braces 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 batch the coding in blocks of 8 abstracts each. Coders pick and process 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 B. (Lutz had coded Block 17 A previously.) Lloyd reserved the block in the coordination file `sample-who-what.txt` and performed the coding.

Step 2. He then ran `check-codings` 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. Compatible means: Differing at most in the subjective suffixes :I and :U, but not in the codes (and 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 correct his coding.

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

3.7.2 Handling Disagreements

Step 5. For the remaining report blocks, Lloyd would write an email to Lutz explaining his reasoning. ((5: What about Step 5a?))

Step 6. Lutz would read through that email and categorize the report blocks into the following cases:

a) Lloyd's coding is obviously correct, Lutz' own is a clerical error. Lutz would correct his coding and respond accordingly. Figure 2 shows such a case.

b) Lutz finds Lloyd's coding clearly incorrect. He would respond with an explanation why he thinks so.

c) Lutz finds Lloyd's coding acceptable, but his own coding preferable. He would respond with an explanation of his reasoning and suggest that Lloyd either adjust his coding or add an -IGNOREDIFF marker to it. Semantically, this suffix indicates two accepted alternative interpretations of the same sentence. Programmatically, it silences the `compare-codings` script for this particular report block, so that the coding difference is now officially accepted.

d) Lutz finds his own coding acceptable, but Lloyds preferable. He would either adjust his coding or add an -IGNOREDIFF marker to it and respond accordingly.

e) 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 very 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 content analyses code most of their material only once, then code a random subsample a second time, compute some coefficient of agreement, report 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 – which is great, because that is what our study is interested in.

Note that, technically speaking, our procedure also leads to a 100% perfect 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

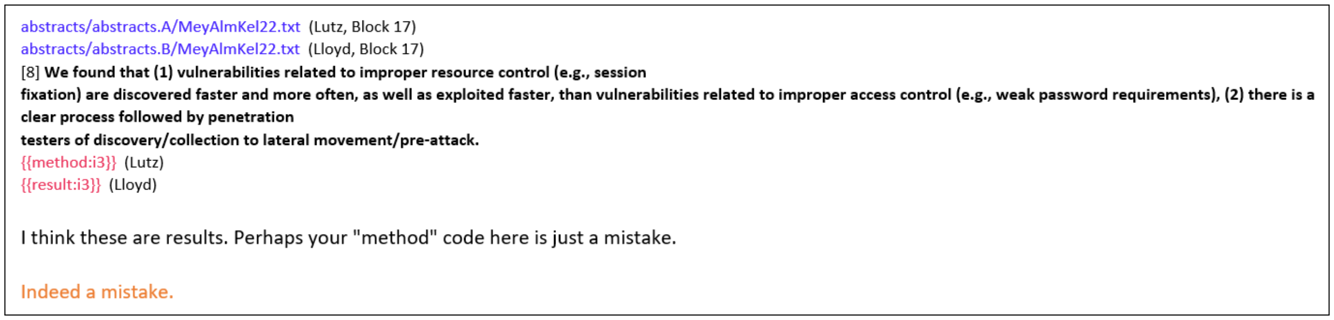
The difficult part of our statistical evaluation is asking the right questions; our data provides a lot of possibilities. In contrast, the actual statistical techniques are simple and straightforward: Mostly tabulations of counts or percentages, bar plots, and box plots. We define two binary properties 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 the results section as making an abstract improper. ((6: add subsection with an evaluation of these two))

We mostly refrain from performing significance tests or computing confidence intervals, because most analyses are exploratory, not driven by specific expectations or theories. Also, most phenomena we report are gradual by nature. The one exception from this rule is the comparison of structured versus unstructured abstracts. ((7: perform significance test))

In this spirit, we often use the following verbal terms for frequencies: rare (less than 5%), not rare (5-20%), common (20-35%), frequent (35-50%), dominant (over 50%).((8: Find spots where these should be used but are not.))

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. ((9: Find spots where such an example is missing.))



abstracts/abstracts.A/MeyAlmKel22.txt (Lutz, Block 17)
abstracts/abstracts.B/MeyAlmKel22.txt (Lloyd, Block 17)
[8] **We found that (1) vulnerabilities related to improper resource control (e.g., session**
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
testers of discovery/collection to lateral movement/pre-attack.
{{method:i3}} (Lutz)
{{result:i3}} (Lloyd)

I think these are results. Perhaps your "method" code here is just a mistake.

Indeed a mistake.

Fig. 2. Excerpt from Lutz' response email during the disagreements handling for Block 17: Report block from the `compare-codings` script at the top, text line from Lloyds first email in the middle, Lutz' response below. This one was a simple case; sometimes several lines of text are provided by each coder. Overall, Lloyd's first email contained report blocks for 9 disagreements in 4 abstracts, both typical numbers.

3.10 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 would be 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 200 to 290 words long and 8 to 14 sentences long. Structured abstracts tend to be longer than unstructured ones. See Figures A.10 to A.12 for details. The official word limits of 150–250 for EMSE, maximum 300 for IST, and recommended maximum 250 for TSE are disobeyed by more than a quarter of all articles for each of these three venues.

The Flesch-Kincaid "reading ease" readability score [21] is a validated and widely used metric that judges readability of English text based only on the number 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".

See Figure 3 for the results: only about 10% of all abstracts have good readability, less than 40% have normal readability, and a majority is improper — almost a quarter is even below 10! ICSE tends to be better than the other venues, TOSEM is worse.

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

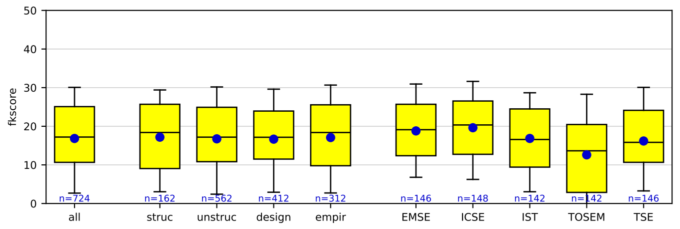


Fig. 3. Flesch-Kincaid 'reading ease' readability score, higher is better. Values under 50 are considered difficult to read (college-level material). Under 30: very difficult to read (graduate level, acceptable for abstracts). Under 10: extremely difficult to read (overly difficult). Differences between subgroups are modest. ICSE is best (mean 20), TOSEM is worst (mean 13). Structured abstracts are just as difficult as nonstructured ones if one ignores the "Methods:" etc. headings as we did here.

on how the authors phrase their goal in their OBJECTIVE statement. Consider the following goal statements:

((10: Find 2 examples))

The first puts the artifact at the center, so this is a design work. The second puts empirical results at the center, so this is an empirical work and the DESIGN code will not be used for them. Artifact design discussion will then usually be coded as METHOD instead. Mixed cases are rare and then the coders will decide which aspect has more weight.

In design works, a large fraction of the abstract will be devoted to describing design considerations for that artifact; in our data: typically 16% to 34%!!! of the words. The empirical study, which usually exists as well, then usually has a mere supporting role (validating the claims made in the artifact discussion) and is correspondingly given less space: 7%–16% (versus 12%–23% for empirical works) for description of empirical method, 12%–24% (versus 18%–32%) for description of empirical results. Design work abstracts are barely longer than empirical work 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 software engineering abstracts archetype:

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 point.
6) Occasionally (but infrequently), *Study Description* will end with a study summary (SUMMARY).
7) 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, in no particular order.

The archetype describes all variants of abstracts that have a natural train of thought, deviations will tend to lead to a less easily understandable abstract.

((11: Find short, good, unstructured example and mark it up.))

4.4 How Not To: Inefficient Allocation of Space

If one reads those abstracts, 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. Here is an example from an article titled "!!!":

((12: Does an example pay off here? It needs a lot of space))

Other abstracts manage to solve the same problem in a wonderfully concise manner:

((13: Find example))

As we can see in the Background group of boxplots in Figure 4, some venues have a quarter of articles that spend one third or more of the abstract space on background. This leads to deficiencies later on, as we can see in Figure 5: 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: Murky OBJECTIVE statements

The OBJECTIVE statement is the key means by with an abstract communicates to the reader what the work is all about. Getting it wrong is therefore probably the single biggest mistake to make for the abstract writer.

We saw a number of near-incomprehensible cases in our sample. For instance ((14: horrible objectives statement example 1)) should better have said ((15: appropriate replacement for example 1)). And ((16: horrible objectives statement example 2)) should better have said ((17: appropriate replacement for example 2)).

4.6 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 often-missing GAP and *Outlook* parts FPOSS, FNEED, etc. are clearly optional, so it is not a problem that they are often not present. BACKGROUND and OBJECTIVE are hardly ever missing.

METHOD and RESULT are sometimes missing (and this is a problem), but never in structured abstracts.

The shocking part of this analysis is CONCLUSION, which ought to be present as the key take-home message in any abstract, but is in fact missing in more than half of all – yet again very rarely in structured abstracts[7]

4.7 How Not To: Convoluted Trains of Thought

Botching the OBJECTIVE statement or leaving out important parts from an abstract are not the only ways for wrecking the abstract's value, however. There are also abstracts where the train-of-thought is so convoluted that they become difficult to read.

For quantifying this, we map each abstract to a short sequence of letters, where each letter stands for a contiguous stretch of sentences in the abstract that have the same content type; the letter designates that content type.

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 content types sequence background, objective, methods, results, conclusion.

The full list of abstracts structures is too long to show it here. It has 95!!! entries for empirical articles and 122!!! entries for design articles. Most of these have only one or two instances, though; we restrict ourselves to the types that occur with higher frequency here. Note that a frequency of e.g. 2.5 simply means that the two coders did not agree. The results are shown in Figure 7 for empirical articles and Figure 8 for design articles. ((18: leave out these plots to save space?))

4.7.1 Empirical Articles

Let us start with the simpler empirical articles. We see that the most sensible structures are also the most frequent (bars 1, 2, 3): Archetypical with or without a GAP statement, perhaps with an *Outlook*. But from bar 4 on, the abstracts are more or less pathological: The first few are mostly missing a conclusion, later on (not shown in the figure) follow all kinds of curious structures such as, to pick an admittedly extreme case, bgomrcsrsc: two conclusions, two summaries, summary after conclusion.

7. Note that the prompt of having to write something after a "Conclusion:" keyword alone cannot guarantee this: The statement put there can be (and sometimes is) something else, typically a RESULT statement or SUMMARY.

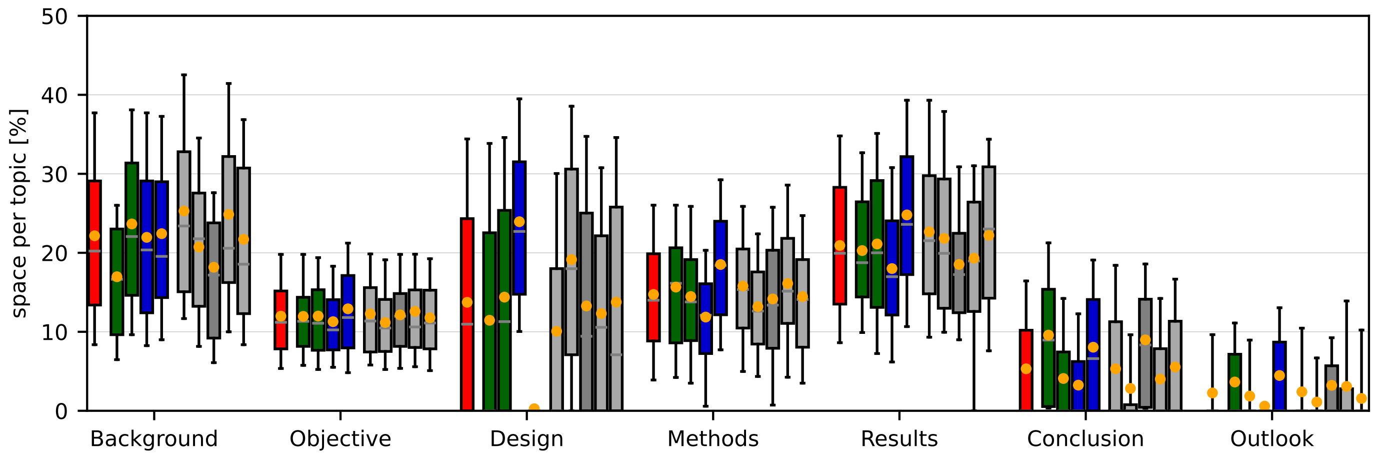


Fig. 4. Per-topic distribution of the amount of space used for that topic. These "topics" are groupings of related codes; e.g. Outlook stands for the union of FPOSS, FNEED, FGAP, FWORK, and all corresponding A-* and (theoretically) H-* codes.
The box shows the 25-to-75 percentile, the whiskers are 10- and 90-percentile, the grey bar is the median, the fat dot is the mean.The plots in each group show these different subsets of abstracts: all, structured, non-structured, design, empirical, EMSE, ICSE, IST, TOSEM, TSE.

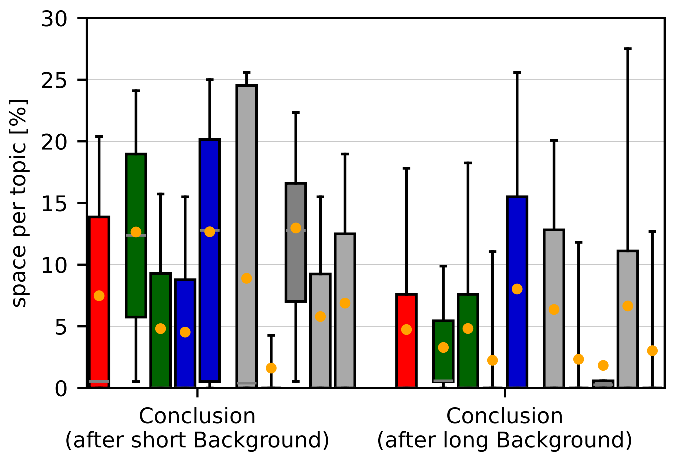


Fig. 5. Comparison of the amounts of remaining space devoted to the conclusion for abstracts with a rather short background section (lowest quarter) vs those with a long one (highest quarter). The latter conclusions are shorter even though the indicated percentage pertains to the part of the abstract after the background only.
The plots in each group show these different subsets of abstracts: all, structured, non-structured, design, empirical, EMSE, ICSE, IST, TOSEM, TSE.

4.7.2 Design Articles

It naturally gets worse for the design articles with their more complicated structure: Even the top two structures are missing a conclusion and we find complex repetitive structures starting at bar 5. Not all of these are automatically bad. In particular, having two small empirical studies, resulting in a nice bgodmrmrc structure, can make a lot of sense for the reader. But this has its limits: An abstract with structure bodmrmrmrmr is overloaded. It can also easily get out of hand, as confirmed by the examples bobdrmrmr and bomdmrmrmrsmrc.

4.7.3 Structured Abstracts

Today's conventions for structured abstracts may be a bit restrictive (e.g. by not accommodating the useful mrmr substructure), but they do result in more orderly abstracts: There are only 27!!! different abstract structures for the structured abstracts of empirical articles versus 95!!! for empirical articles overall. ((19: replace 95 by mean of same-sized samples))

4.8 How Not To: Uninformative Formulations

A milder 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.

4.8.1 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 appear mostly in results statements (82% of the gaps) or method statements (16% of the gaps). Most (if not all) of them could be filled by a number. They tend to cluster, like here: *"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."* [ShiBiaBri22] 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 have an informativeness gap (making them improper) and 18%!!! have three or more. See the leftmost two groups of Figure 9 for details.

4.8.2 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

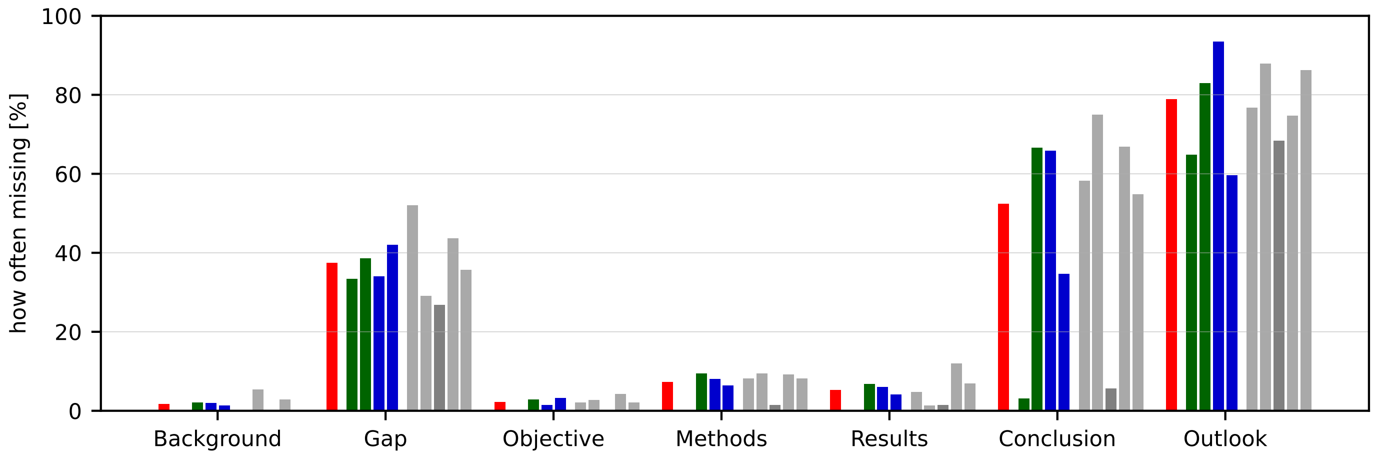


Fig. 6. How often is a topic not present at all in an abstract?
The plots in each group show these different subsets of abstracts: all, structured, non-structured, design, empirical, EMSE, ICSE, IST, TOSEM, TSE.

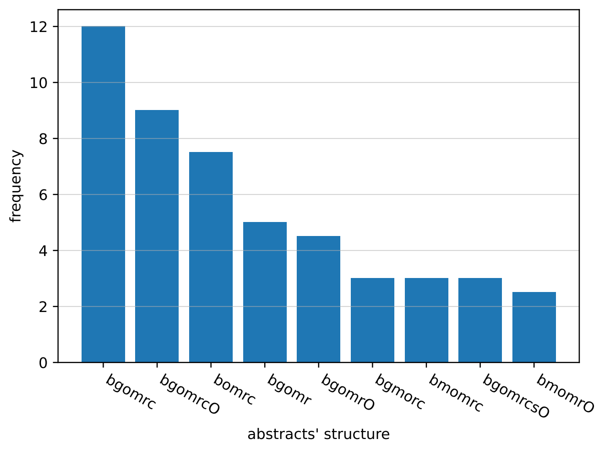


Fig. 7. The frequency of different trains-of-thought in the abstract for empirical articles. The label is a string of stretch-code characters: b-ackground, g-ap, o-bjective, d-esign, m-ethod, r-esult, s-ummary, c-onclusion, O-utlook.

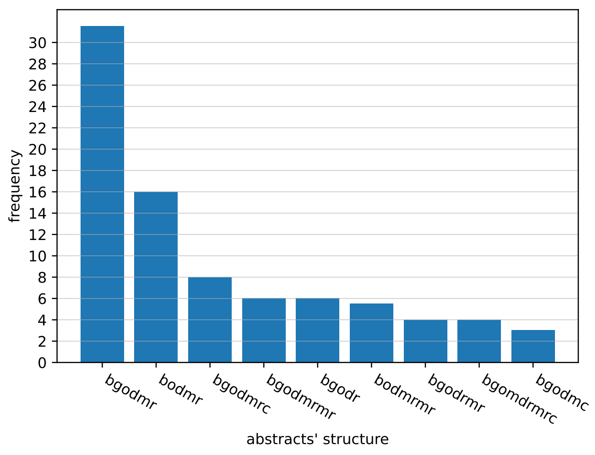


Fig. 8. The frequency of different trains-of-thought in the abstract for design articles. Same characters as before, except that d can now in fact occur.

be found in the article body. We call such a sentence an announcement. ((20: Should we cross-reference back to the respective methods section here? And elsewhere as well?))

Example 1 (method announcement): *"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."* [BesMarBos22] 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 2 (results announcement): *"Finally we provide guidelines/best practices for researchers utilizing time-based data from Git repositories."* [FliChaDye22] 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% of all abstracts have at least one and we consider those improper. See groups three to six of Figure 9 for details.

4.9 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 becomes so large that the sentence containing it becomes incomprehensible. We call such a term use an understandability gap and consider such abstracts to be improper.

Here is an example: *"The results of evaluating the generality of the iContractML 2.0 reference model show that it is 91.7% lucid and 72.2% laconic."* [HamMetQas22] Neither of the terms "lucid" or "laconic" have been introduced before, so that this sentence has two understandability gaps and an average

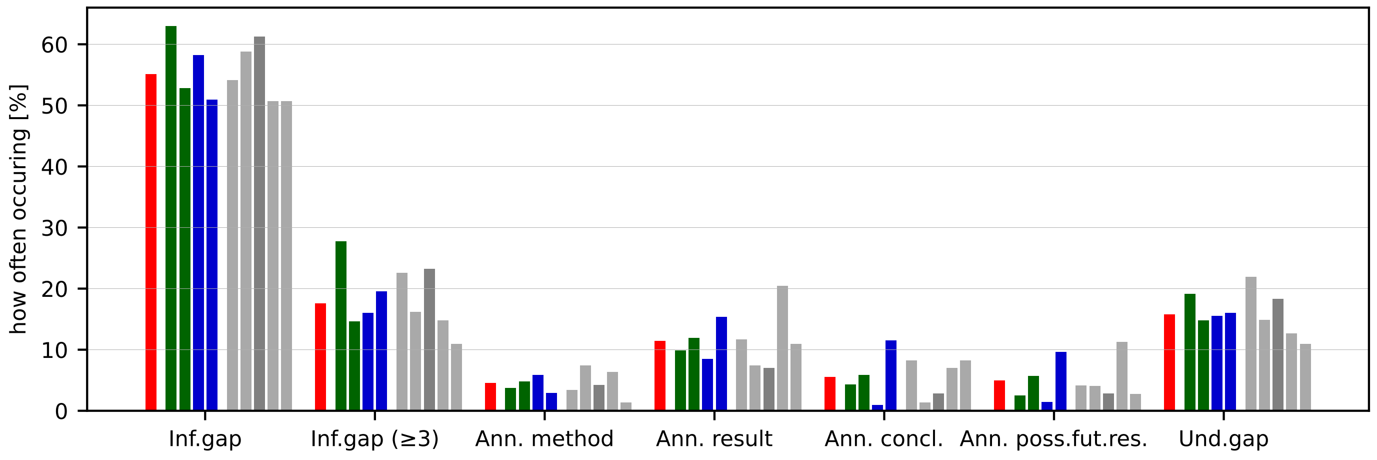


Fig. 9. What fraction of abstracts has the following uninformative types of formulations?
One-or-more or three-or-more informativeness gaps (i.e. missed opportunities for being more specific).
Some sentence that only announces (instead of describing) a method, result, conclusion, or possible future research.
One or more understandability gaps.
Informativeness gaps are epidemic, worse(!) for structured abstracts. Announcing is generally not rare, in particular for results, and tends to be less pronounced at IST. Not explaining key terms is not rare and worst at EMSE.
The plots in each group show these different subsets of abstracts: all, structured, non-structured, design, empirical, EMSE, ICSE, IST, TOSEM, TSE.

reader is not able to make sense of this statement which represents half of the work's results.

See the rightmost group of Figure 9 for how frequent this is: About 17%!!! of all abstracts have one or more understandability gaps. The issue is most pronounced at ICSE.

4.10 How Not To: Ambiguous Formulations

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

In our data, we found 74!!! such cases overall, spread over 30!!! different abstracts, so that 5%!!! of all abstracts have one or more such problems. A total of 14!!! different codes are involved.

4.11 How-Not-To Summary

((21: implement 'complete' and 'proper' (and its parts) in printstats.))

Summing up, a proper abstract should ((22: collect the properties)). Given that we have looked at what are usually considered top-quality venues, one should expect that a large majority of abstracts fulfills those criteria. However, only x% have property1, only x% have property2, only x% have property3, [...]. And only x% have all of these properties. Only x% are at least complete in the sense that they describe background, objective, methods, results, and formulate a conclusion.

4.12 How To: Structured Abstracts are more orderly

((23: ditto for structured abstracts, probably as a side-by-side plot of the partial criteria.))

5 Limitations and Threats to Validity

5.1 Interpretivist Perspective

In terms of Tracy's quality criteria for qualitative research [22], 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 better abstracts.

6 Conclusions

6.1 Too Few Abstracts Have Good Quality

((24: spell out)) Only !!!% are even complete, only !!!% also fulfill modest additional criteria: !!!list them!!! and this despite us analyzing only presumably high quality venues.

We conclude 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 4.12). Take care to avoid announcements (see Section 4.8.2), understandability gaps (see Section 4.9), and sentences with an unclear role (see Section 4.10). Provide helpful detail, perhaps simplified, if it consumes barely any space (see Section 4.8.1).
2) Write a BACKGROUND section that provides just enough context and motivation to understand the subsequent OBJECTIVE (see Section 4.4).
3) Decide on your main contribution. Is your article a design article or an empirical one? Write an OBJECTIVE that expresses both succinctly (see Sections 3.4.2 and 4.5). If your background information contains a corresponding GAP statement, mark it as such.
4) If your work is a design article, write a DESIGN section. Cover all key ideas (typically two to four), avoid non-key information.
5) If you have multiple near-independent empirical sub-studies 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 4.7). Otherwise write separate METHODS and RESULTS sections as follows.
6) 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 4.8.1).
7) 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 4.8.2).
8) 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 relevance and good enough credibility. Expect to later be proven wrong for some of your works, but not for many (see Section 4.4).
9) An outlook on future possibilities *can* be part of your CONCLUSION, but is usually better left to the body of your article.

((25: Take a good short design work abstract and show it in this format?))

6.2.2 For Venues

Structured abstracts are not currently used widely in software engineering. In their usual form, which does not suit design articles, this is understandable, even appropriate.

In the above extended form, however, (allowing GAP statements, allowing DESIGN sections, allowing multiple METHODS AND RESULTS sections) structured abstracts promise better abstracts quality than a free-style format. Therefore, venues should require structured abstracts in this engineering-ready format.

For your call for papers, feel free to copy the above text and either remove the "see Section" cross references or point authors to this article for the underlying evidence.

Acknowledgments

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

References

[1] T. A. Lang, "Scientific abstracts: Texts, contexts, and subtexts," *European Science Editing*, vol. 48, 2022.
[2] A. Strauss and J. Corbin, *Basics of qualitative research*. Sage Publications, 1990.
[3] K. Krippendorff, *Content analysis: An introduction to its methodology*, 2nd ed. Sage Publications, 2004.
[4] J. M. Swales, *Genre analysis: English in academic and research settings*. Cambridge University Press, 1990.
[5] M. B. Dos Santos, "The textual organization of research paper abstracts in applied linguistics," *Text*, vol. 16, no. 4, pp. 481–500, 1996.
[6] C. Cross and C. Oppenheim, "A genre analysis of scientific abstracts," *Journal of Documentation*, vol. 62, no. 4, pp. 428–446, 2006.
[7] 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.
[8] 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.
[9] 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.
[10] 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.
[11] C. 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

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

[19] 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.

[20] 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.

[21] J. P. Kincaid, R. P. F. 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," 1975. [Online]. Available: https://stars.library.ucf.edu/istlibrary/56/

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

Appendix

This appendix contains figures that provide additional detail for information that is only summarized in the article proper.

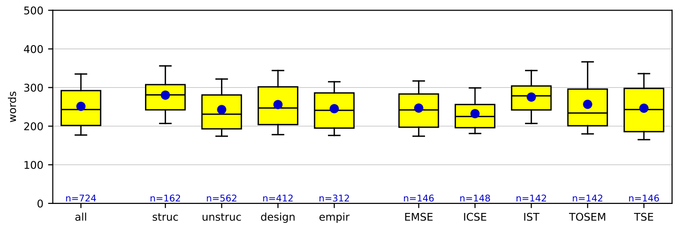


Fig. A.10. Typical abstracts are 190 to 280 words long. Structured abstracts tend to be longer than unstructured ones.

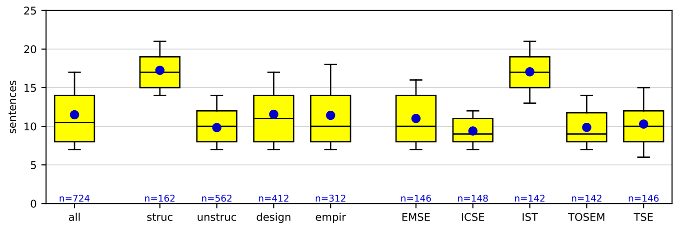


Fig. A.11. Typical abstracts are 8 to 14 sentences long. The values for structured abstracts are inflated by the section headings, which each count as a sentence if they terminate in a colon.

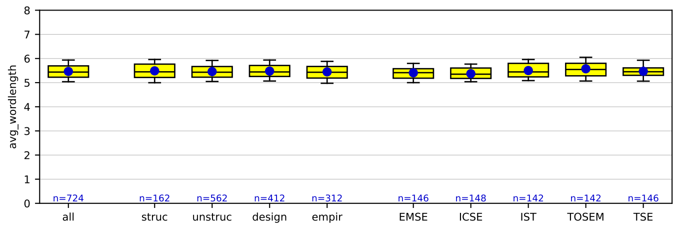


Fig. A.12. The average number of characters in a word is similar in all subgroups.

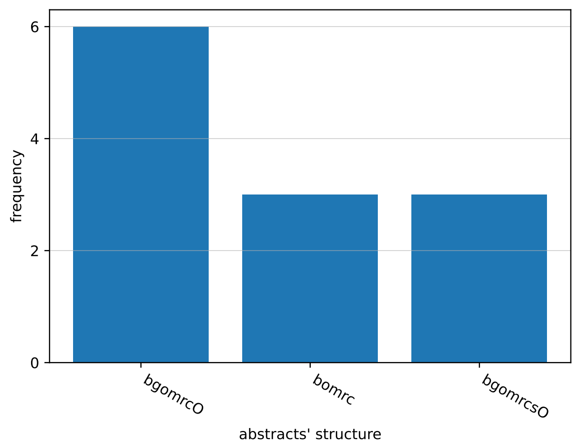


Fig. A.13. The frequency of different trains-of-thought in the abstract for empirical articles using a structured abstract format. The "Method:" etc. subheadings are ignored for structured abstracts in all three plots.

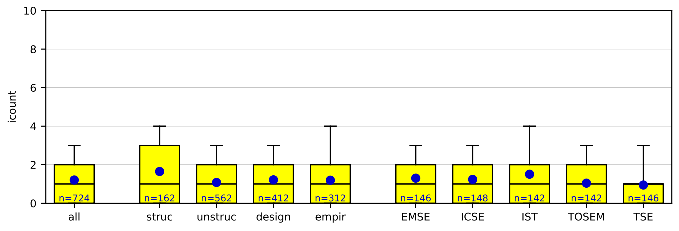


Fig. A.14. A majority of abstracts has one or more "informativeness gapst't'. The problem tends to be smaller in structured abstracts.

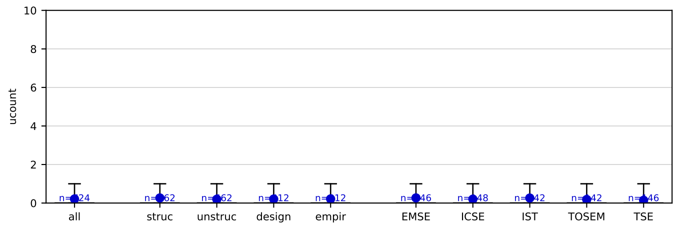


Fig. A.15. Some abstracts have understandability gaps where an important concept is not explained at all. !!!

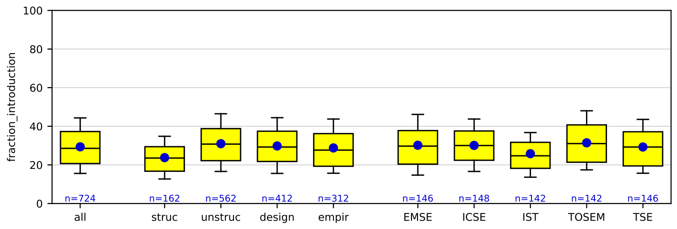


Fig. A.16. Authors invest about a third of the space into the introduction: background/context information and perhaps explaining a research gap. Structured abstracts tend to have less of this???

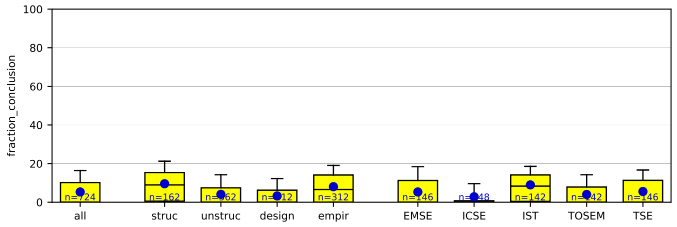


Fig. A.17. Many abstracts offer no conclusion, i.e., no explicit generalization of the immediate result. This is worst at ICSE (with over 75%) and best at IST, which is the only venue where a majority of articles formulates a generalization.

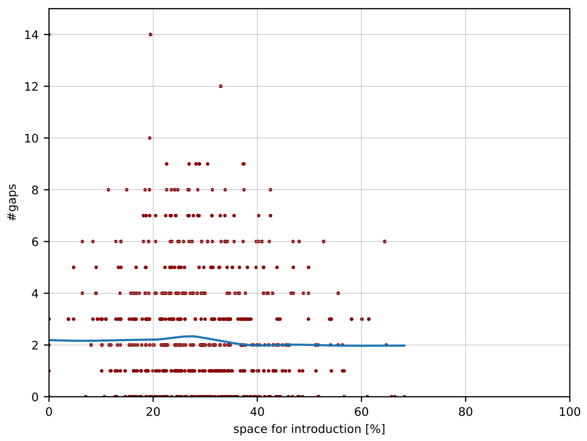


Fig. A.18. This plot checks the expectation that spending much space on background will correlate with more i and u gaps. This is not the case. Each dot is one coding of an abstract, the line is a locally weighted linear regression.