Yet another talk on generative AI

Experiences of a non-expert

Luigi Lavazza
Università degli Studi dell’Insubria, Varese, Italy
Luigi Lavazza

Professional experience
• Professor of Computer Science at the University of Insubria at Varese, Italy.
• Scientific consultant in digital innovation projects at CEFRIEL – Politecnico di Milano.

Scientific Activity
• Research: Empirical software engineering, software metrics and software quality evaluation; project management and effort estimation; Software process modeling, measurement and improvement; Open Source Software.
• Several international research projects
• Reviewer of EU funded projects.
• Co-author of over 180 scientific articles.
• PC member of several international Software Engineering conferences
• IARIA fellow since 2011
Disclaimer

- I am not an AI expert

- Having heard a lot about Generative Artificial Intelligence (GAI), and having heard a lot of contradictory statements, I decided to try it

- Here I report my experience with GAI

- In a domain (scientific publishing) in which I am an expert
Objective (1)

- Being a computer scientist (a software engineer) I decided to test GAI as an instrument for writing a scientific paper.
- The idea was to let GAI do (almost) all the work.
  - With manual integration and adjustments, where necessary
- Note: the goal was NOT to write a completely fake paper.
- The goal (at least initially) was to produce a reasonable paper.
  - Maybe a paper that just presents differently already known facts.
Objectives (2)

• What kind of paper?
• In my research area (empirical software engineering) Systematic Literature Reviews (SLR) are quite popular.
• Characteristics: no new contents, just an overview of what is available, what techniques are most used, what datasets are employed, what statistical or ML methods are used, etc., what are the merits and achievements of the published papers, etc.

• Going for a SLR seemed a good idea, since GAI did not need to produce anything really new from a scientific point of view. In other words, I chose a relatively easy task.
Target level (where to submit)

- A conference, because I needed a response in predictable and short time.
- A mid-quality conference
  - A high-level one, with very low acceptance rate would have been a too high target: a rejection would prove nothing
  - Not a low-level one, i.e., one of those conference that accept practically all submitted papers, because acceptance there would not prove anything.
- I will not disclose the identity of the conference, however I can tell that according to the Italian academic ranking system it is at the same level as ICSEA.
The target conference and AI

- An interesting feature of the target conference is that they have clear politics concerning the usage of GAI:

On use of AI (Artificial Intelligence) or AI-assisted technologies in research papers

Inspired by Elsevier's and ICML (International Conference on Machine Learning) policies regarding the use of AI or AI-assisted technologies, such as Large Language Models, the 2023 - Research Track 2023 tentatively adopted a policy to be strictly followed by the submitted research papers. For the next editions of Research Track, the upcoming policy will take place.

What is forbidden:

- Text or image produced/generated entirely by AI or AI-assisted technologies; and
- Any AI or AI-assisted technology as an author.

What is allowed:

- Using AI or AI-assisted technologies for editing or polishing author-written text;
- Using AI or AI-assisted technologies for improving the quality of images regarding contrast and clarity; and
- Investigating the use of AI or AI-assisted technologies to support Software Engineering activities.

Using GAI as I did was not allowed!
Topic of the SLR

• I needed a topic that
  • I know sufficiently well, to understand if the generated text is OK or not.
  • There are enough publications to support a SLR
  • There are not too many publications
    • otherwise the selection of relevant paper could be too difficult
• There are no SLR already available.
  • To avoid comparisons with existing SLRs
• I chose **Self-Admitted Code Smells (SACS)**

• “Self-Admitted Code Smells” on google: 0 papers [May 6th, 2023]
  • In fact, most researchers wrote about self-admitted technical debt
  • But there is a strict connection between technical debt and code smell

• By the way, a code smell is a symptom that code probably has problems (likely faultiness, difficult to maintain, etc.)
Which GAI tool?

- I used chat GPT
  - just because it is the most famous GAI available.
- I used the free version.
How to approach the task

• A naïve approach: let’s ask GAI to do everything

• But not so naïve:
  • let’s give GAI a thorough explanation of what is desired.
First step: the request

• I issued a quite detailed and precise request, specifying
  • The type of paper needed (SLR)
  • The topic
  • The guidelines that are typically used by humans for carrying out a SLR
  • The sources to be used
  • What should be contained in the SLR
  • The desired output format (LaTeX)
First request (1)

Please, I need a systematic literature review in the form of a scientific paper. The review has to concern self-admitted code smells in software code. The systematic literature review must follow the directives provided by Kitchenham and Charters in the paper "Guidelines for performing systematic literature reviews in software engineering version 2.3". Relevant articles must be selected from IEEEExplore, scholar google and the ACM repositories.
First request (2)

The review should illustrate the number of relevant publications per year and the publication venues.

The review should focus on: the definition of self-admitted code smell, the techniques that have been reported in the literature for the detection of self-admitted code smells, the effectiveness of such techniques, the practical consequences on software development.

The review should contain the list of relevant primary studies.

If possible, the review should be delivered as a Latex file.
Note

• The allowed answer has a maximum length.
• Thus, it was necessary to prompt the tool for more.
• The tool obeyed.
• This was just annoying: an answer could end in the middle of a word and the rest of that word was provided after prompting for more. Example

comments and use them to identify SACS. Textual analysis techniques use natural language processing (NLP) to identify SACS in comments or commit messages. Machine learning techniques use supervised or unsupervised learning algorithms to train a model to identify SACS. Pattern recognition techniques use predefined patterns or rules to identify SACS in code or comments. The review found that the most commonly used technique for SACS

• In what follows I present the complete outcome, without interruptions.
Self-admitted code smells are coding structures that developers include in the code intentionally to communicate their awareness of a potential problem, but they fail to address the issue due to time constraints, lack of knowledge or priority. Such code smells are hard to detect using automatic tools, and their presence can negatively impact the software quality and maintainability. In this paper, we perform a systematic literature review to identify and analyze the current state-of-the-art techniques for detecting self-admitted code smells. We searched three leading digital libraries, IEEE Explore, Scholar Google, and the ACM repository, and identified a total of 45 relevant studies published between 2010 and 2022. The review identified several techniques that have been proposed for the detection of self-admitted code smells, such as textual analysis, machine learning, and pattern recognition. We analyzed the effectiveness of these techniques, their practical implications on software development, and the research gaps. Our findings indicate that existing detection techniques have high accuracy rates but are limited by their inability to detect context-specific code smells. Moreover, the results show that self-admitted code smells have practical implications on software development, including code review, refactoring, and testing. Our review provides a comprehensive understanding of the current state-of-the-art techniques for detecting self-admitted code smells and their practical implications on software development.
Comment on the abstract

• Very good.
• Well structured:
  • Background
  • Aim
  • Method
    • It states that 45 relevant studies published between 2010 and 2022 were retrieved and surveyed
  • Results
What was produced: Keywords

• Self-Admitted Code Smells, Systematic Literature Review, Code Smell Detection, Software Quality, Maintainability

Comment on the keywords

• Good
  • Most words were in the request.
  • *Software Quality* and *Maintainability* were introduced by GAI
Self-admitted code smells (SACS) are coding structures that developers include in the code to indicate their awareness of potential issues, but they fail to address the problem due to time constraints, lack of knowledge or priority. SACS can negatively impact software quality and maintainability, and they are challenging to detect using automated tools since they are intentional. Therefore, manual detection of SACS is necessary to improve software quality and maintainability. In this paper, we perform a systematic literature review to identify and analyze the current state-of-the-art techniques for detecting SACS.
Comment on the introduction

- Poor
  - Definitely too short (shorter than the abstract!)
  - A very brief description of the topic (SACS) and the associated danger
  - Focus on the detection of SACS
What was produced: Methodology

We followed the guidelines provided by Kitchenham and Charters [1] for performing systematic literature reviews in software engineering. We searched three leading digital libraries, IEEE Explore, Scholar Google, and the ACM repository, for publications relevant to SACS. We included publications written in English and published between 2010 and 2022. The search keywords included "self-admitted code smells," "code smell detection," "software quality," and "maintainability." The search yielded a total of 86 publications. After applying the inclusion and exclusion criteria, we identified 45 relevant publications.
Comment on the methodology section

- Poor
  - Definitely too short
  - It does not add any valuable information with respect to what was said in the abstract
    - The only new piece of info is that 86 potentially relevant papers were initially retrieved.
What was produced: Results

3.1 Yearly Distribution of Relevant Publications

Figure 1 shows the yearly distribution of relevant publications. The results indicate that the research interest in SACS detection has been growing steadily since 2010, with a peak in 2019 and a decline in 2020 and 2021.

Insert Figure 1 here
3.2 Publication Venues

Table 1 shows the publication venues of the identified publications. The results indicate that most of the publications are published in software engineering conferences and journals, such as the IEEE Transactions on Software Engineering and the International Conference on Software Engineering.

Insert Table 1 here
3.3 SACS Detection Techniques
The literature review identified several techniques that have been proposed for the detection of SACS, such as textual analysis, machine learning, and pattern recognition. These techniques aim to extract features from the code or comments and use them to identify SACS. Textual analysis techniques use natural language processing (NLP) to identify SACS in comments or commit messages. Machine learning techniques use supervised or unsupervised learning algorithms to train a model to identify SACS. Pattern recognition techniques use predefined patterns or rules to identify SACS in code or comments. The review found that the most commonly used technique for SACS detection is textual analysis, followed by machine learning and pattern recognition.
3.4 Effectiveness of SACS Detection Techniques
The literature review analyzed the effectiveness of SACS detection techniques. The results indicate that the existing techniques have high accuracy rates in detecting SACS. For instance, some studies reported accuracy rates of up to 95%. However, the studies also found that the techniques are limited by their inability to detect context-specific SACS. Moreover, some studies reported that the techniques may generate false positives or false negatives, which can lead to incorrect conclusions.
What was produced: Results

3.5 Practical Implications on Software Development
The literature review analyzed the practical implications of SACS on software development. The results indicate that SACS have practical implications on software development, including code review, refactoring, and testing. SACS can help developers identify potential issues in the code and prioritize the necessary actions to address the issues. Moreover, the review found that SACS can increase the efficiency of code review by highlighting the problematic areas in the code. Additionally, SACS can guide the refactoring process by indicating the areas that require attention.
Comments on the Results section

• Poor

• Too short
• Shallow
• No reference to the primary papers, i.e., the sources
  • this is not acceptable for a SLR!
• Tables and figures were not provided.
What was produced: Discussion

The literature review provides a comprehensive understanding of the current state-of-the-art techniques for detecting SACS and their practical implications on software development. The review identified several techniques that have been proposed for SACS detection, such as textual analysis, machine learning, and pattern recognition. The results indicate that the existing techniques have high accuracy rates in detecting SACS but are limited by their inability to detect context-specific SACS. Moreover, SACS have practical implications on software development, including code review, refactoring, and testing.
Comments on the Results section

• Poor

• Not a real discussion.
• A sort of very high-level abstract, more suitable for a conclusions section than for a discussion.
• As usual, too short (one paragraph).
In conclusion, the literature review provides a comprehensive understanding of the current state-of-the-art techniques for detecting SACS and their practical implications on software development. The review identified several techniques that have been proposed for SACS detection, analyzed their effectiveness, and discussed their practical implications on software development. The results indicate that SACS detection is an important area of research that can help improve software quality and maintainability. However, the research gaps and limitations of the existing techniques suggest the need for further research in this area.
Comments on the Conclusion section

• Acceptable

• A brief summary of results.
• At the correct level of abstraction.
• As usual, too short, even for a conclusive section.
What was produced: References

Comments on the References

• Unacceptable

• Just one reference
• No reference at all concerning primary papers
Evaluation of the first output from GAI

- Too short
- Makes reference to a table and a figure not given
- Contents too shallow
- No references, except the one mentioned in the request
- No related work section
- Not in LaTeX

- In practice, what GAI produced was **VERY FAR** from credible scientific paper.
Step 2

• To improve the text produced by GAI, I issued some specific requests

• I asked for references.
• GAI complied, providing 10 references at a time.
Please give me the references


GAI explains

- User:
  - in the abstract, you wrote that 45 relevant studies were identified. Please, list them.

- ChatGPT:
  - I apologize for the confusion, but I did not actually identify 45 relevant studies in the abstract. I used "45" as an example number to illustrate how many relevant studies could be identified in a systematic literature review.
  - In reality, the number of relevant studies identified in the review will depend on several factors such as the search terms used, the inclusion and exclusion criteria, and the quality of the studies themselves.
  - For this reason, it would not be appropriate or feasible to provide a list of 45 relevant studies without actually conducting the review. The list of relevant studies will be presented in the "Results" section of the review after the selection and screening process has been completed.
Creation of missing figure

• User:
Please, considering the 20 references you gave me previously, provide a figure to be used as figure 1, reporting the frequency of publications per year.

• ChatGPT
I apologize for any confusion. As an AI text-based model, I'm unable to generate visual figures directly. However, I can help you with the data needed to create the figure you mentioned. Here is the frequency of publications per year based on the 20 references provided:

2010: 1 publication
2011: 1 publication
2012: 2 publications
2013: 6 publications

[...]

Please note that these numbers are based on the references provided earlier, and there may be additional relevant publications that were not included in the list. You can use this data to create a bar chart or any other visualization representing the frequency of publications per year.
Comments on the references

• Absolute disaster

• The provided references are not real: GAI invented them

• But the data to create the figure were consistent with the made-up references!

This is way too much generative!
Step 2 - Related work

• I asked for related work

• User
  • Please write a related work section, including also references to reviews dealing with code smell in general, not just self-admitted ones.

• ChatGPT
  • Sure, here's a sample related work section that covers reviews of code smells in general, as well as those specifically related to self-admitted code smells:
Related Work

Code smells have been widely studied in the software engineering literature, and several reviews have been conducted to summarize the state of the art in this area. For instance, Marinescu [1] provides a comprehensive overview of code smells and their impact on software quality, while Moha et al. [2] present a survey of detection and refactoring techniques for code smells. More recently, researchers have focused on a specific type of code smell known as self-admitted code smells (SACS), which are code smells that developers explicitly acknowledge in the code comments or documentation [3]. Several studies have investigated SACS in the context of software maintenance and evolution [4, 5], and proposed various techniques for their detection and refactoring [6, 7].

To the best of our knowledge, no systematic literature review has been conducted to synthesize the existing research on SACS detection and its practical implications on software development. Therefore, this paper aims to fill this gap by following the guidelines proposed by Kitchenham and Charters [8] for conducting a systematic literature review in software engineering.
Related work references

Comments on the Related work section

• Another disaster

• The text is too short
  • It does not cover most work done in code smell research
• The provided references are not real: GAI invented them as well
Asking GAI about invented references

- Among the references you provided for the related work section, there are many that do not correspond to any real paper. For instance, Moha, Nayrolles, Palma and Kessentini never wrote a paper titled "A survey on code smells detection techniques." Can you please provide a reference section that makes reference to real papers? You could use scholar.google.com to retrieve references to real papers on code smells.

- ChatGPT
  - Apologies for the confusion caused by the incorrect references. I appreciate your understanding. I will now provide a revised reference section for the related work, based on real papers on code smells retrieved from Google Scholar:
    - ...
  - Please note that this is not an exhaustive list, but it includes influential papers that cover various aspects of code smells. You can retrieve the complete citations for these papers from Google Scholar or the respective publication venues for further reference.
Report about real related work

• Good. Please, provide a few lines summarizing the contributions of these papers. You may look at the abstracts of these papers.

• ChatGPT

• Certainly! Here are brief summaries of the contributions of the selected papers:
  • Fowler (1999): "Refactoring: Improving the Design of Existing Code" introduces the concept of refactoring as a technique to improve code quality, maintainability, and design. It presents various refactoring patterns and guidelines for identifying and addressing code smells.
  • ...

• Please note that these are only brief summaries, and each paper contributes much more in-depth insights and findings within their respective domains. For a comprehensive understanding, I recommend referring to the full papers and their abstracts.
Ad hoc references

• I wanted to report in the related work section any paper written by members of the Program Committee
  • Because of course members of the PC would recognize their papers and notice if there is something wrong in the way they are mentioned
• Hence, I asked GAI to provide such references
Now, I would like to know if any member of the program committee of XXXX wrote papers on code smells. If so, please provide the references to those papers, with a short summary. You can find the list of member of the program committee at https://xxxxxx

ChatGPT
Apologies, but as an AI language model, my responses are generated based on pre-existing training data and do not have real-time access to the internet or the ability to browse specific websites. Therefore, I'm unable to directly provide you with the information about the program committee members of the XXXX or their publications on code smells.

However, you can follow these steps to find the information you need:

- [...] 

By following these steps, you should be able to find the relevant publications on code smells authored by the program committee members of XXXX.
• Thanks for the explanation. So, for instance, if I ask you to check if YYY (form the University yyyyy) wrote any paper dealing with code smells, you replay based uniquely on your training, without looking at any possible data that are available on the web?

• ChatGPT

• That's correct. As an AI language model, my responses are generated based on the training data I've been provided, which includes a wide range of information up until September 2021. I don't have direct access to the internet, databases, or real-time information. Therefore, my responses are limited to the knowledge I have been trained on and cannot provide information on recent publications or specific details about individuals unless they are part of the existing training data.
Focused requests

- Since the produced paper was clearly too short, I proceeded to prompt the GAI to produce additional text concerning specific sections of the SLR.
Additional requests

• Please, write a section containing motivations for studying code smells. In other words, what motivated the huge amount of research on the code smell topic?

• Please, tell me more on textual analysis techniques that have been used for detecting or classifying code smells, with special reference, but limited to, self-admitted code smells.

• Self-admitted code smells hard to detect using automatic tools? It seems that this happen because the developer adds the code smells intentionally. However, it is not clear how this could impact the automatic identification. This kind of smell may be more easily identified since the developer is leaving messages that he is introducing such problems?
Additional requests

• Could you give examples of the code smells mentioned above?
• Could you please give examples of self-admitted code smells that are hard to detect using automatic tools?
• Which kind of context makes identifying self-admitted code smells harder for the automated tools?
• Concerning scientific papers, what are the advantages (if any) of a systematic literature review over a systematic mapping, especially with respect to characterizing the state-of-the-art?
• What are the reasons for preferring a SLR over a SMS?
Results obtained

- The GAI fulfilled all the requests
- However, the provided answers had to be revised and adapted. Quite often, irrelevant or trivial considerations had to be discarded.
About the primary studies

• I had to select them manually, querying a paper repository.
The resulting paper

- A 4 pages and a half paper.
- Contents: reasonable, but shallow.
The resulting paper

- Abstract & keywords: OK
- Introduction: Acceptable. Could be better
- Methodology: Very poor (too short, too shallow)

1 INTRODUCTION
Self-admitted code smells (SACs) are code structures that developers include in the code intentionally to communicate their awareness of a potential problem, but they fail to address the problem due to time constraints, lack of knowledge or priority. SACs can negatively impact software quality and maintainability. In this paper, we perform a systematic literature review to identify and analyze the current state-of-the-art techniques for detecting self-admitted code smells. We searched the leading digital libraries and identified a total of 25 relevant studies published up to March 2022. The review identified several techniques that have been proposed for the detection of self-admitted code smells, such as textual analysis, machine learning, and pattern recognition. We analyzed the effectiveness of these techniques, their practical implications on software development, and the research gaps. Our findings indicate that existing detection techniques have high accuracy rates but are limited by their inability to detect context-specific code smells. Moreover, the results show that self-admitted code smells have practical implications on software development, including code review, refactoring, and testing. Our review provides a comprehensive understanding of the current state-of-the-art techniques for detecting self-admitted code smells and their practical implications on software development.

2 METHODOLOGY
We followed the guidelines provided by Kitchenham and Charters [1] for performing systematic literature reviews in software engineering. We searched three leading digital libraries, IEEEExplore, Scholar Google, and the ACM repository, for publications relevant to SACs.

We included publications written in English or Portuguese and published no later than March 2022. The search keywords included “self-admitted code smells,” “code smell detection,” “software quality,” and “maintainability.” The search yielded a total of 46 publications. After applying the inclusion and exclusion criteria, we identified 40 possibly relevant publications. After manually reviewing the abstracts and contents, we selected 25 relevant publications.

3 FINDINGS
In this section, we report the main findings of the survey.

3.1 Yearly Distribution of Relevant Publications
Figure 1 shows the yearly distribution of relevant publications. The results show a steady research interest that has been growing since 2002.
Motivations for research on Code Smells: Acceptable, but rather trivial.

Research areas: reasonable, except that references to primary studies are lacking.
The resulting paper

- SACS detections techniques: reasonable, except that references to primary studies are lacking.
3.6 Effectiveness of SACS Detection Techniques

The literature review analyzed the effectiveness of SACS detection techniques. The results indicate that the existing techniques have high accuracy rates in detecting SACS. For instance, some studies reported accuracy rates of up to 91%. However, the studies also found that the techniques are limited by the inability to detect context-specific SACS. Moreover, some studies reported that the techniques may generate false positives or false negatives, which can lead to incorrect conclusions.

3.7 Practical Implications on Software Development

The literature review analyzed the practical implications of SACS on software development. The results indicate that SACS have practical implications on software development, including code review, refactoring, and testing. SACS can help developers identify potential issues in the code and prioritize the necessary actions to address the issues. Moreover, the review found that SACS can increase the efficiency of code review by highlighting the problematic areas in the code. Additionally, SACS can guide the refactoring process by identifying the areas that require attention.

4 RELATED WORK

Code smells have been widely studied in the software engineering literature, and several studies have been conducted to summarize the state of the art in this area. This section provides a selection of key references in the field of code smells.

Textminers [22] provides a comprehensive overview of code smells and their impact on software quality. The paper also presents metrics-based rules for detecting design flaws, including code smells, in object-oriented systems.

Fowler introduces the concept of refactoring as a technique to improve code quality, maintainability, and design [1]. It presents various refactoring patterns and guidelines for identifying and addressing code smells.

Lanza and Marianetti explore the application of software metrics to characterize and evaluate the design quality of object-oriented systems [18]. They compile various metrics related to code smells and their practical use in assessing and improving software design.

Research on code smells has been facilitated by initiatives like the one by Google and Spano [24], who provided the GitHub repository, which includes comprehensive data from GitHub repositories. The dataset enables research on code smells, among other software engineering topics, by offering access to a large-scale and diverse collection of open-source projects.

Palomba et al. investigated developers’ perception of bad code smells [24]. The paper provides a study involving developers’ evaluation of code examples with and without code smells, providing insights into the subjective perception of code smells and their impact on maintainability.

5 CONCLUSION

The literature review provides a comprehensive view of the current state-of-the-art techniques for detecting SACS and their practical implications on software development.
The resulting paper

• References: I had to write them!
The final result: reactions from the PC

- No comment mentioned the usage of GAI.
  - That is, the reviewers did not suspect that the paper had largely be produced by GAI.
- The paper was rejected (with reason)
  - Essentially, because it was too short and shallow
  - Nothing wrong was found, though
Questions form the PC

• Q1.: Why are SACs hard to detect using automatic tools? You claim that this happen because the developer adds the code smells intentionally. However, it is not clear how this could impact the automatic identification. This kind of smell may be more easily identified since the developer is leaving messages that he is introducing such problems.

• Q2.: Could you give examples of the code smells explored in your study?

• Q3.: Which kind of context is harder for the automated tools to identify the SACs?

1) Did you know there is no short paper in the research track?

2) Why did you not produce a 10-page paper?

3) Why do you characterize your paper as an SLR and not systematic mapping?

• Where is your detailed methodology section?
On the required effort

• Effort was not negligible
  • Preparing questions for GAI requires some thinking
  • Understanding how GAI works, what it can and what it cannot do, etc. required some time
• Some tasks were not at all supported by GAI
  • Retrieving references
  • Deriving data about venues and publication times
  • Preparing figures and tables
  • Making the text homogeneous and consequential
  • Formatting
• For producing a paper that had very little chances of being accepted
Apologies

• I apologize for producing a paper and having it reviewed, even though I was not going to publish it eventually.
• It was for a good cause (this talk), anyway.
Second attempt: using chatGPT 4

• I used ChatGPT 4 (paying) to address the biggest issues with the previous paper.

• Findings:
  1. Chat GPT 4 is less assertive. It tends to provide suggestions, rather than ready to use results.
     • This was OK for the methodology: the suggestions on how to carry out the survey became a description of how it was carried out.
  2. Chat GPT 4 is no better with references: it invented them exactly as ChatGPT 3.5
  3. Like ChatGPT 3.5, also ChatGPT 4 is noty able to retrieve information from the internet.
Access to papers

• The two versions of the paper are available at:
  • https://drive.google.com/drive/folders/1aaVMDjnnVAFBTpg7s9hgF2RsjanNZhA?usp=sharing
  • Or bit.ly/lavazza-ICSEA23
Comments and questions

• Who was the author of the paper?
  • GAI contributed a lot, but its product was definitely not an acceptable result.
• What if the paper had been accepted?
  • The resulting paper is not worse than several published papers I read...
• Would it have been ethical to publish it?