Essay: Software Engineering for AI/AS

Lukas Eveborn
Linköping University
lukas.eveborn@liu.se

August 20, 2025

1 Introduction

My research area is within mathematical optimization or more narrowly combinatorial optimization. The topic of my research is the development of algorithms for solving large-scale optimization problems, particularly focusing on routing problems for fleets of electric vehicles. A stripped down version of the problem, could be described as follows: Given a fleet of vehicles starting at a depot, some goods need to be delivered to set of customers. You should then to decide which customers should be visited by which vehicle and in what order, while minimizing the total cost of the routes. This while also ensuring that limitations such as vehicle capacity is respected. This and other routing problems is not something new, it has been around for quite some time and is quite well studied, but the introduction of electric vehicles adds a number of complex aspects to it, such as charging infrastructure and battery limitations.

In order to solve realistically sized problems it is not possible today to just write down a mathematical model and then use a black-box solver to find the optimal solution. Instead, specialized (and fast) algorithms are needed, often combined in a quite delicate manner. This leads to a situation where no matter the problem that we want to solve, quite some code/software is needed to be used and developed, both for the algorithmic components and for the integration with the surrounding systems. In my academic setting, the software for the surrounding setting is typically not in focus, rather the emphasis is on the optimization algorithms themselves. Everything else "should just work", which is far from my understanding of how it is in the industry, where the surrounding part often is as important, or even more so, than the core algorithms.

2 Lecture Principles

The first principle or concept that sort of stuck with me was the second finding from the case study on code review done at Google [5] that was discussed. It was about that the expectations on a code review depend on the work relationship between the one giving the feedback and the one receiving it. It does not only have to be close colleagues whose goal is to find issues in the code, but also others that are not directly involved in the development, but maybe with another purpose. This principle or concept is not be directly related to my research itself and what is seen as the contribution, but since a big part of what I do to get to results is coding, it is close to the research process. In most cases, there is no code review

done at all today, since well there are no other "developers" in the team, or in other words no one which could find issues with the code with a reasonable effort. However, if I try to think about it a bit more like in the finding I think there could be quite some value, at well-chosen points, doing code reviews. First and foremost maybe my supervisor - in order for her to get a better understanding of what I'm doing - which may lead to better discussions.

The second principle is related to Behavioral Software Engineering and the main reason why it exists: People are not rational. The logical consequence and actual principle is then that it is not enough to only focus on better technology - many times it can be more effective to focus on the individuals/groups instead. I think this is spot on for my research area, as often, given my understanding, it is not the algorithms that are the bottleneck in practical use, but rather getting people to use them as intended. One concrete example I have heard many times, is the behavior of truck drivers which when given a route to follow it quite often happens that they do not follow it. The reasons could be many, but one is that software simply did not plan their lunch-breaks at their favorite spots along the route. If we take a step back and focus on planners using the software instead, there can be similar problems. Planners may have their own mental models of how a good route should look like, which may not align with the software's suggestions. One quite nice solution to this I heard from a company was to make it easy for the planners modify and play around with the suggested routes in the software, something which over time made them more comfortable with the software and its suggestions.

3 Guest-Lecture Principles

The first one I'm focusing one is the principle of Cost of Defect Removal based on [2] and [1] which Julian presented. What I found interesting is that the cost of removing defects increases by roughly a factor of 10 for each step in the 5-step process, starting from requirements engineering and finishing with deployment. Leading to the take-away that you want to catch defects as early as possible in the process. I do not know if the exact same factors apply to my research, but I think the principle is still valid and can quite easy be translated to my work. The requirements in my case is the general research idea, the architecture is how it should be done algorithmically, implementation and testing stays the same and the deployment is the final result, which might be a paper. For example throwing away a bad idea early in the process is much "cheaper" than finding out that the idea is bad months later while writing the paper.

The other guest principle is also from Julian's lecture in which he stated *"Every project has requirements - but not every team decides to write them down"*. And, connected to this said - in contrast to what I have always been taught - it is not necessary (in all cases) to write down requirements in a formal way, what is important is that there is a shared understanding of the requirements. I found that quite liberating, as I often have felt that writing down requirements would be a bit of a waste of time, especially when it comes to software development I do in my research. However, I do agree and think it is important even to my research to actually think about the requirements and what I want to achieve, but it does not always have to be in a formal way, when it is between only me and my supervisor.

4 Data Scientists versus Software Engineers

In broad terms, I agree with what is stated in chapter 1 and 2 in the CMU "Machine Learning in Production" book [3] regarding the essential differences between data scientists and software engineers. They have different skill sets and focus areas, where data scientist is more focused on the data and the algorithms, while software engineers are more focused on the software and the system. However, I do not think that these differences are as big or clear-cut as the book suggests in most cases, and I believe in many cases there are an overlap between the two roles. A data scientist do most likely have quite a bit of knowledge about software engineering and probably do some software engineering tasks as well. The other way around I think also holds true in many cases, but maybe not always to the same extent.

Moving forward, I have hard time predict how these two roles will evolve, I think further specialization is possible as well as a more of a blending of the two roles. Arguing for the case of further specialization, I would say this would, let's say, a natural continuation of what I see as long-term trend both in software but also in society, where specialists are more and more valued. On the other hand, blending of the two roles would be logical from the perspective of system thinking - the ML-model is just part of the system and it is important to consider the entire system when developing it and vice versa. If you believe their conjecture *"Our conjecture is that we tend to attempt much more ambitious and risky projects with machine learning."*, a more blending or persons that have a better understanding of both roles becomes more important to be able to understand and handle the risk in a better way.

5 Paper Analysis

This section provides an analysis of two papers from CAIN conferences, for which I ended up choosing *Unmasking Data Secrets: An Empirical Investigation into Data Smells and Their Impact on Data Quality* [4] from the 2024 edition and *Investigating Issues that Lead to Code Technical Debt in Machine Learning Systems* [6] from the 2025 edition.

5.1 Paper 1: Unmasking Data Secrets: An Empirical Investigation into Data Smells and Their Impact on Data Quality

For this paper, the authors tried to answer three questions, firstly what data smells are already documented and which are the tools to detect these? Secondly, in real-world datasets, how often do these data smells occur? Thirdly, how do these data smells impact the quality of the data? For the first part they defined three new categories of data smells, in addition to what already existed in the literature; *Redundant Value Smell*, *Distribution Smells* and *Miscellaneous Smells* as well as listing a number of tools for detecting data smells. In the second part, analysing a total of 19 datasets, they found that 14 of them contained at least one data smell. The by far most common was *Extreme Value Smell* and *Missing Value Smell*. *Causing Smell* and *Suspect Sign Smell* also appeared in several datasets, but not to the same extent. For the third part they found that individual data smell might have a small impact on the data quality, but the cumulative effect can have a significant impact.

I think the ideas/findings in this paper are important to the engineering of AI systems, as the data is an

important part of the system. They showed that data smells can effect the quality of data, that smells do exists in real-world datasets. And of course in order to be able to find them, you need to now which types of smells that exists. As to the relation to my research, I need to handle a lot of data, as input to the optimization algorithms. Sometimes it is academic data-sets, but there are also cases where I need to work with real-world data, and even though this data typically is not used to building AI-models, I would assume (or sometimes I know) that these datsets have data-smells as well that effect the data quality.

One AI/ML intensive project where the ideas from the paper could be used, is in route planning for electric vehicles where one important but also difficult aspect is to accurately estimate the energy consumption for a vehicle on different road segments. The reason is that a lot of factors influence the energy consumption, such as the road gradient, the speed limit, the weather and the type vehicle. One way to estimate the energy consumption is to use machine learning models trained on historical data, to try to model the energy consumption. This would then be the input to the optimization algorithm, outputting the best routes for the vehicles. It would a project close to my WASP research as I today focus on the optimization algorithms, but at the moment typically use either academic datasets or static models/data as input to the optimization. It would also be important to consider the potential data smells in the input data and how they could impact the quality of the optimization results, because you wouldn't want to end up with routes that might not even be feasible in reality because of poor data quality.

One potential way I could use the ideas from the paper in my research, especially if I would start with something more like described above, would be to start to use some of the tools to try to detect data smells in the datasets I use. By that hopefully being able to address the data smells before they become a problem. Another possibility, which would be more of a research project, could be to investigate how data smells in the input data impact the optimization results.

5.2 Paper 2: Investigating Issues that Lead to Code Technical Debt in Machine Learning Systems

This second paper tries to identify and discuss code-issues that can lead to technical debt (TD) in machine learning systems (ML-systems). They end up with a list of 30 issues out of which 24 are deemed to be critical. These are them divided among the seven typical phases of an ML-workflow, showing that the majority of the issues are related to the *data pre-processing* phase. The *data collection* and *model creation and training* phases also have some issues, while the remaining have none or very few.

As with the first paper, I think the ideas/findings in this paper are important to the engineering of AI systems, because if you want to build something more than just a prototype, you need to be able to maintain and develop it (the code) further. TD in software engineering is well-known and something that can cause major issues, so to think that they do not exist in ML-systems would be naive. And of course, to be able to address and be able to work with issues that can cause, it helps to be aware what they can be in the first place. As to the relation to my research, when I look at the issues identified in the paper, especially connected to the *data collection* and *data pre-processing* phase I can see quite some parallels to my own work where I work with quite some data. However, today I might just try to fix many of these from case-to-case or just ignore them, but especially if it should become something more robust or scalable, these issues would probably cause some TD along the road.

Regarding an AI/ML intensive project where the ideas from the paper could be used, I think the same

project as for the first paper could is a good example. In connection to this paper, the focus would be on trying to focus on the code development aspects and how to ensure that the code remains maintainable and does not accumulate technical debt over time. As they pointed out in the paper, it is especially important to think about the issues in the step of *data pre-processing* but also *data collection* and *model creation and training* as they were identified as the phases with the most issues.

One way I could tweak my research project to better support the ideas from the paper, would be to try to focus more on the code development aspect, especially when it comes to the data collection and pre-processing. One aspect in this would be to trying to move from excel sheets and manual data handling to more automated and code-driven approaches.

6 Research Ethics & Synthesis Reflection

My screening process started by reading all the titles from the three last years, 2025, 2024, and 2023. Based on the titles, I then read the abstracts of the six papers that I found most interesting and for which I understood the title. Out of those I selected three papers for a more in-depth read to start out with. After that I picked the two of them for the analysis, even though quality wise all would have been fine.

In general, I think the titles quite good represented/summarized the content of the abstract, and in the next step the abstracts did a good job of summarizing the content of the papers. The fact that I only picked papers for which I understood title I think reduced the risk of being misled by the title or abstract. One could argue that I might have missed some interesting papers but given the content and quality of the papers I ended up reading, I think this was a good-enough approach in this case.

For the third point, I am not completely sure I understand what is meant, but I assume it is about not copying the sources or using LLM:s to write the text. Regarding that, I did not ask any LLM to write the text and then just copying it. However, I wrote my text in VSCode with Copilot active, which means that it did suggest some text snippets to finish sentences when I started writing. Most of the time I did not use these, but of course I was influenced by them from time to time.

References

[1] Barry W Boehm. Software engineering economics. *IEEE transactions on Software Engineering*, (1):4–21, 1984.

[2] D Méndez Fernández, Stefan Wagner, Marcos Kalinowski, Michael Felderer, Priscilla Mafra, Antonio Vetrò, Tayana Conte, M-T Christiansson, Des Greer, Casper Lassenius, et al. Naming the pain in requirements engineering: Contemporary problems, causes, and effects in practice. *Empirical software engineering*, 22(5):2298–2338, 2017.

[3] Christian Kaestner and Eunsuk Kumara. *Machine Learning in Production*. Carnegie Mellon University, 2024. Accessed: August 20, 2025.

[4] Gilberto Recupito, Raimondo Rapacciuolo, Dario Di Nucci, and Fabio Palomba. Unmasking data secrets: An empirical investigation into data smells and their impact on data quality. In *2024*

IEEE/ACM 3rd International Conference on AI Engineering – Software Engineering for AI (CAIN), pages 53–63, 2024.

[5] Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko, and Alberto Bacchelli. Modern code review: a case study at google. In *Proceedings of the 40th international conference on software engineering: Software engineering in practice*, pages 181–190, 2018.

[6] Rodrigo Ximenes, Antonio Pedro Santos Alves, Tatiana Escovedo, Rodrigo Spinola, and Marcos Kalinowski. Investigating issues that lead to code technical debt in machine learning systems, 2025.