Occam's Razor in CS: Creating Value for Clients in the Simplest Way Possible

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science
University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science, School of Engineering

Thomas Mullins Arnold

Spring, 2025

Technical Project Team Members
Thomas Arnold

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Rosanne Vrugtman, Department of Computer Science

Occam's Razor in CS: Creating Value for Clients in the Simplest Way Possible

CS4991 Capstone Report, 2024

Thomas Arnold
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
ybn4aq@virginia.edu

ABSTRACT
A major manufacturer with a major market share in its industry found its manufacturers and client-facing account managers working without any communication, causing unfulfilled orders and frustration for the end customer. To solve this problem, my intern team and I created a full-stack inventory management and communication system using the MERN (MongoDB, Express, React, and Node) tech stack. This allowed plant managers and account managers to be more in sync with each other to provide the end customer with maximum transparency. Despite our manager's request to incorporate artificial intelligence (AI), I convinced the team not to use AI because our solution did not need AI and it would only slow us down. Ultimately, ours was one of the only solutions actually deployed by the end of the internship, and our approach allowed us to meet all of our stretch and scrum goals throughout the summer. My conclusion: In an ever-changing computer science world that includes AI, often the best solution is still just the simplest one. I would like to test this theory further by doing more client projects that avoid time-intensive and/or complex technologies during development and compare the two for deliverable hits.

1. INTRODUCTION

The term "Tutorial Hell," infamous in the world of computer science, refers to the ease with which novice programmers get caught up in the process of learning a new language, methodology or technology instead of spending valuable time creating software. Computer science is a daunting profession to get into, since there is a plethora of programming languages, development environments and application programming interfaces (APIs), as well as an infinite number of project ideas and a vast array of ways to approach given project ideas, along with seemingly limitless combinations of tools.

There is a lot of pressure on new programmers to be up-to-date with the latest technologies. The problem with this is that there are always hot new technologies being created and iterated upon. This pressure makes sense, as programmers would understandably want to be fluent in hot technologies to give themselves a better chance at finding work or creating deliverables so that stakeholders can be awed by how "high tech" they are.

I argue that pushback is needed. The computer science world has lost its way by favoring overly complex and resource-intensive technologies in lieu of attacking a computational problem in the simplest way possible. Often these complicated approaches actually result in a sharp decrease in performance.

2. RELATED WORKS

Gonsalves, et. al. (2023) interviews Ali Shojaei, who discusses how using AI can save precious development time, especially on relatively menial tasks like referring to documentation or debugging. Shojaei go on to say that using AI during development can similarly help eliminate human error. However, he also suggests that merely using AI as a "silver bullet solution" for the sake of satisfying stakeholders can lead to poor results in the end deliverable if developers do not have the necessary knowledge. I would go a step further to say that even if the developers are knowledgeable in the field of AI, incorporating AI as a buzzword in a project is hazardous, and the time spent incorporating the technology would be better invested in just making the core features of the project without AI (Gonsalves, et al, 2023).

Netflix's UI/UX team (2017) delivered a keynote in October 2017 that explained how switching from React, a very popular web development framework, to simple, vanilla JavaScript saw a 50% decrease in their Time to Interactive (TTI) metric, which measures how fast a webpage loads. I believe this is a perfect example of how using simple tools can result in better results than using complex alternatives. React is a framework that is always in high demand due to its popularity around the technology world. However, learning React and conforming web design choices to it requires a large time commitment. I would argue, with support from the Netflix report, that using the simplest tools can often reduce bloat and create a better user experience (Netflix UI Engineering, 2017).

3. PROCESS DESIGN

This process compares and contrasts software built with complex technologies to those built with simple technologies. I argue that simple software is easier to develop, manage, and deploy, and that the benefits of simple software development will be made evident to anyone who subscribes to this belief.

3.1 Defense

The software of our modern world is overwhelmingly complicated. It is almost unfathomable just how many users use software from companies like Amazon, Apple, and Microsoft. A litany of issues come up when referring to software of this scale: How do we keep this software secure? How do we scale up our software to accommodate more users? How do we ensure that the massive computational load is distributed efficiently among servers? How do we onboard new software engineers and get them accustomed to our system?

In response to these questions, many companies opt to use the latest software development frameworks like React, Vue, and now, AI. This response makes sense, as companies are always looking for ways to optimize their development pipeline and use hot technologies in order to attract investors. For that matter, a lot of what software companies do is to appease investors, as is evident with the rise of AI. Every major tech company is now including some form of artificial intelligence in their end products, for better or for worse.

I argue that, in terms of making quality, maintainable software, we should treat direction from investors more lightly. Often, investors are not very knowledgeable in software development. Their job is to look at the market trends and put cash into companies that have a chance to grow. They are usually not the people actually designing the software they are investing in, so their advice on what technologies to use should not be taken as seriously as they are now. When companies do fold to investor demands and

include complex technologies like AI into their deliverables, the results can be mixed and unpredictable. For example, Google has recently adapted their generative AI service Gemini into their titular search engine. Adding a technology that is very prone to hallucinations to a service that people use to get reliable, truthful information has resulted in a decrease in quality of the Google search engine. When an internet user typed into Google "I'm feeling depressed," Gemini suggested that the user "[jump] off the Golden Gate Bridge." When asked to name African countries that begin with the letter k, Gemini confidently stated that no such country exists. Pairing this with the fact that running a Gemini search query uses 10 times as much water for cooling as a regular Google search, one begins to ponder why Google even chose to adopt AI in the first place.

Using more complex technologies, frameworks and APIs in a codebase necessarily requires more code to be written. Although there is no direct function to relate bugs and lines of code, it is undeniable that more code naturally leads to more bugs and security vulnerabilities appearing in software. Especially when the code being written deals with third party tools, you are at the mercy of those developers to ensure that your own code which relies on those tools runs smoothly. If one is to use a third party API in their program, at any point that API could come offline or be introduced to a security vulnerability, possibly crashing the original software or otherwise compromising it.

Introducing more third-party APIs into a code base also makes maintenance substantially more difficult. Not only do new hires or internal code checkers have to learn the ins and outs of the homemade software, but they must learn the third-party technology as well, which eats away at precious development time. If a blank project starts with leveraging a substantial number of third party APIs, the initial development time is dragged, as all the developers will have to learn this new foreign technology. Writing more code also comes with a time and space tradeoff–the code files themselves will take up more space and use more resources to run.

These are not just theoretical hits–decreases in software efficiency directly cost a company more money to operate and maintain the software. Of course, it is impossible to expect software to not use some third-party tools and technologies, especially software deployed on a massive scale. I am not arguing against the use of third-party technologies, or even against the use of AI in all cases. It is undeniable that tools like React have revolutionized software development, and OpenAI's ChatGPT has changed the way we interact with computers. However, we must remember that the goal of software development is to develop efficient software that satisfies stakeholders' needs–sometimes adopting these tools can hurt the quality of the software and/or the software development cycle.

3.2 Evaluation

To evaluate how effective simple software is at hitting deliverables, one only has to look at the quality of the end deliverable and how many key points were hit. For example, if a manager asks a team of software engineers to design a video streaming website, one can easily fathom how to measure the quality of the end product: Can the site stream videos? Is it laggy on certain devices? Is the UI accessible? Measuring the simplicity of software is more difficult. However, I generally define simple software as software that avoids using as many third-party technologies, APIs, frameworks, and tools as possible.

How do engineers know that their solution is simple? Unfortunately, there is no formulaic way to answer this question; it all depends on the context of the development of the software. In the video streaming example, one could examine how many third party APIs were used. Then, one could consider how necessary each of those APIs were: Could the requirements satisfied by these APIs have been developed in house? If so, how much more or less time and money would it have taken to do so?

A great way to evaluate my simple software theory is to ask two teams of engineers to develop a product, instructing one of them to use as little third party tools as possible, and applying no such restriction on the other. Then, one would compare how many deliverables were hit and the quality of the two approaches. To test this theory specifically as it pertains to AI, one could tell the first team to include AI in the final deliverable and prohibit the other from doing so. Naturally, the development cycle of the first team will be longer, but one still only needs to evaluate the end deliverables of the two teams to see differences in quality.

4. EXPECTED RESULTS

Adopting this simple approach to software engineering could revolutionize the software world. By eliminating time spent learning and testing new tools, teams can directly develop the software in meaningful ways. This will make working on large, established code bases more accessible for new software engineers, as there will be less to learn before they can write code. I believe that AI is unnecessary in most software applications that use it. It is expensive, both in terms of money and resources needed to power the systems, and it adds a massive roadblock to the development of the software.

5. CONCLUSION

I implore tech companies to adopt my philosophy about AI, as AI is known to be unreliable and hallucinate information anyways. For example, search engines should opt not to use AI, as people are looking for true information, not information regurgitated from an AI model that uses significant freshwater to power it. Of course, if this methodology is adopted, the companies that make these third party tools and AI models will suffer, as fewer people would use them. To this, I argue that markets drive innovation, and if people are using companies' tools less and less, the company will be more inclined to iterate upon and improve the design of their tools, increasing the quality of future software that does use them.

6. FUTURE WORK

An expanded verison of testing this theory involves a company or software team adopting the simple, AI-less approach for an extended period of time, perhaps a year. During this time, end users would be surveyed on the quality of the software, how likely they are to continue using the software, and how likely they are to recommend to the software to a friend. Then, after the company or team has allowed themselves to produce simple software for this period of time, the survey data will be examined and contrasted with previous user satisfication data. If a company is doing this, they could look at the earnings report or stakeholder satisfaction during this period and contrast it with relevant data before this period of time. This would give the engineers and stakeholders the clearest assessment on the effectiveness of simple software.

REFERENCES

Gonsalves, F., Green, J., Parrish, A., Moxley, T., Seeber, C., & Williamson, A. (2023, Fall). *AI—The good, the bad, and the scary*. Virginia Tech Engineer.

https://eng.vt.edu/content/eng_vt_edu/en/magazine/stories/fall-2023/ai.html

Netflix UI Engineering [@NetflixUIE]. (2017, October 26). *Removing client-side React.js (but keeping it on the server) resulted in a 50% performance improvement on our landing page https://t.co/vM7JhWhYKu* [Tweet]. Twitter. https://x.com/NetflixUIE/status/923374215041912833