Organizational Structures and Vulnerabilities in the Software Industry

A Research Paper submitted to the Department of Engineering and Society

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

August Diamond

Spring 2024

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

Advisor

Rider W. Foley, Department of Engineering and Society

Introduction

In recent decades, the demand for computer software has risen massively. Humans now turn to computers for things that would have been inconceivable a mere generation ago – from hosting millions of concurrent real-time video conversations during a global pandemic (Karl et al., 2021), to large language models being used to write articles (Hosseini et al., 2023). The percentage of United States residents who own a computer has expanded from approximately 20% three decades ago to 92% as of 2018 (Martin, 2021), with software becoming a ubiquitous part of many millions of lives along the way. However, our increasing reliance on software brings about a new of set problems. Software tools are prone to failure, and the resulting vulnerabilities in programs we trust with our personal, medical, financial, or otherwise sensitive information can have disastrous consequences for individuals and communities at large. With increasing responsibility, the production of high-quality software, absent of major vulnerabilities, is more important than ever.

The increasing demand for complex software has also led to organizational changes in corporations developing software. Companies like Meta and Alphabet, which collectively employ tens of thousands of developers, could not have existed in prior decades, much less stood among the most valuable companies in the world. Now, the top executives of companies like these are some of the wealthiest people on Earth. Speculation on further growth in software & technology companies is of major interest to investors and venture capital, with over $100 billion from private equity entering the systems software industry over the last three years (Sabater & Asif, 2023). Some have expressed worry that the increasing corporatization of software development groups is detrimental to the development of high-quality software, as the monetary interests of non-technical stakeholders, who typically rank above engineers in the corporate

hierarchies, may be prioritized over safety and performance concerns. Recent case studies include the executive suite of Unity, who infamously received a collective ~$100 million in compensation in 2022, making them among the highest paid tech executives in America, in a year where their company suffered costly software quality issues in its advertisement monetization tools and lost 75% of its value (Starks, 2023).

This paper aims to determine the correlation between the types of social organizational structures in which software products are developed, and the frequency of vulnerabilities within these products.

STS Theory

The significance of software quality assurance experts is often poorly understood, even within the software industry itself (Florea et al., 2023). But neglecting the quality assurance process can have significant consequences. In the U.S. alone, it was estimated that nationally, issues with poor-quality software have grown to cost at least $2.41 trillion as of 2022 (Krasner, 2022). In the worst cases, improperly tested software costs not money, but lives. Four years ago, two Boeing-737 Max crashes resulted in 346 deaths after a software component on the planes failed (Ethiopian Civil Aviation Authority, 2022). While software itself was most directly responsible for the accident, there were many social factors at play that led to its failure. The company shareholders and higher-ups, in this case the Boeing executives, desire for growth at minimal cost led to inexperienced, underpaid coders developing crucial parts of their system (Robison, 2019). The users, in this case the pilots, desired a smoother flying experience, which Boeing's new software was intended to provide. And indirect users affected by the system without directly using it, i.e., passengers, want flights to be quick, cheap, safe, and perhaps above

all, frequent. Boeing's own analysis of the growing demand for commercial airplanes (Bergman, 2018) likely contributed to their decision to rush out more products. I find this case study to exemplify some of the common stakeholder archetypes that others have identified in the realm of corporate software organizations (owners, higher-ups, engineers, and the external users), the oft-conflicting desires between them (Kroeger et al., 2014), and the disastrous results of failing to properly mediate their relationships with and influence over software artifacts.

In industrial software development, different stakeholder groups like software engineers and executives exist in a corporate hierarchy that is largely reflected through salary – it is often the highly-compensated managerial groups that have the most power over the direction of the project, even if they themselves are not responsible for creating or maintaining that project on a technical level. A very different organizational structure exists within open-source software communities. In an open-source project, all code is freely available online to anyone who wants to access it, with maintenance and development performed by software engineers who are typically community volunteers, rather than formal members of some distinct project-associated organization. As a result, few open-source projects involve social structures that resemble corporate hierarchies.

Three open-source governance models are particularly common. First is the *benevolent dictatorship*, in which the founding developers of a project make the final decisions on its direction and whether to accept any particular code contribution. *Delegated governance* involves an elected council of community members democratically making these same decisions, perhaps with subcommittees to handle different sets of issues. And finally, *meritocracy* is organized around the community making decisions as a whole, but with more voting power or influence given to developers who have made more substantive contributions (Topelson et al., 2017). In all

of these cases, it is developers that both create and direct the projects. The role of purely managerial or executive stakeholders, monetary compensation, and other mainstays of corporate software development are greatly diminished, if not entirely absent.

Social construction of technology, or SCOT, (Pinch & Bijker, 1984) provides a social constructivist framework for analyzing how human actions shape the creation and development of technology. SCOT can be employed to consider how the security of software is influenced by the organizational structure that governs interactions between its involved stakeholders. When polled, quality assurance-focused developers unsurprisingly tend to prioritize quality over speed and cost (Katalon et al., 2023), while executives may find the latter factors more important. In a corporate hierarchy, the executives' influence may win out, while this would not be the case in any form of open-source organizational structure.

Differences in perspective such as these between involved parties is the result of *interpretive flexibility*, through which the artifact of software takes on a different meaning to different stakeholders. From the perspective of owners and executives, the end-goal of software is often to make the company money, or (to shift one level of abstraction away), to please shareholders. This aligns with the views expressed in the influential Friedman doctrine, a business ethics framework which proclaims that a corporate organization's primary objective should be to act in its own self-interest as an entity, maximizing revenue & shareholder returns, as shareholders are the only group which corporations are socially responsible to (Friedman, 1970).

To developers, on the other hand, creating software is a profession that delivering high-quality code helps them maintain. Developers in corporations are thus directly responsible to higher-ranking employees, including executives. And in many software engineering ethical

frameworks, they are also responsible to their users, the public, and other social groups that are not directly tied to their product's financial success (Gotternbarn et al., 1997). Interpretive flexibility suggests that these values are reflected in the products that executives and software developers produce, and by extension, that different types of organizations will produce software with different qualities, based on how the organization mediates stakeholder relationships.

Case Context

Analyzing corporate software quality is challenging for a multitude of reasons. Perhaps the greatest immediate obstacle to research on this topic is that most corporate software is proprietary. Proprietary, or closed-source software does not have its source code made publicly available, as opposed to open-source code which anyone can access and inspect.

Without access to source code, analysis of software quality can only be done indirectly, through analyzing reports of problems that were found to occur in the proprietary software's use. There are many sources from which one could gather reports. Many news articles have been written about the largest software failures, which can be useful as case studies, but the reporting often lacks technical detail and ignores smaller, yet still significant quality issues. Users frequently take to online public forums to discuss these smaller problems, but it is often difficult to distinguish between issues related to software quality and those caused by other factors (e.g. user error or out-of-date hardware) without inspecting every post individually.

Perhaps the largest and most credible public collection of technically oriented software issue reports is the Common Vulnerabilities and Exposures (CVE) database, operated in part by the U.S. National Cybersecurity FFRDC. The CVE database lists over 215,000 computer security flaws identified in various software applications. This database only records the subset

of software quality issues classified as vulnerabilities, or weaknesses that can be exploited in software programs to cause some form of damage. This does not typically include graphical errors, non-functional issues (e.g. an application taking up too many resources due to poor optimization), or other common software quality issues that cannot be exploited. The database is a strong resource, but it is important to note that with its limited scope, its data can only be used to draw conclusions about software vulnerabilities, rather than the broader topic of software quality issues.

Each entry in the CVE database is accompanied by a score on the Common Vulnerability Scoring System (CVSS), a ten-point standardized indicator of vulnerability severity. The CVSS score for each entry is calculated based on metrics like the exploit's attack vector (e.g. requires physical access to the target system, or able to be performed over a network), what privileges it requires, and how strongly it impacts the confidentiality, integrity, and availability of the targeted system (CVSS V4.0 Specification Document, n.d.). It is worth noting that the CVSS has been criticized for being subject to bias, with scorers bringing their own notions of a vulnerability's impact that may lead them to assign different severity levels from one another, but it is still regarded as a useful tool for vulnerability prioritization among professionals (Wunder et al., 2023).

A final note on CVE entries is that while all types of software are scored with the same system, CVE/CVSS comparisons between different types of software are rarely meaningful. Applications that tend to be less complex or have fewer possible attack vectors will naturally tend to receive fewer, lower-scoring CVEs. More meaningful comparisons can be made between programs that serve the same purpose. Of the types of software in the CVE database, the following stand out as having both several competing products, and a substantial number of

entries per product: databases (e.g. MongoDB, Oracle Database), web browsers (e.g. Google Chrome, Mozilla Firefox), virtual machines (e.g. VirtualBox, Vmware), operating systems (e.g. Windows, iOS), and office tools (e.g. PowerPoint, Adobe Acrobat Reader).

Research Question and Methods

With computers influencing more human lives than ever before, it becomes pertinent to ask: How is the organizational structure of software development groups related to software vulnerability severity? To answer this question, I compiled a list of proprietary software products with entries in the CVE database from the following categories: databases, web browsers, virtual machines, operating systems, and office tools.

Table 1. List of software products selected for analysis (Source: Diamond 2024).

Organization Name	Product Name	Product Type	Open Source?
IBM	IBM Db2	Database	No
Microsoft	Microsoft SQL Server	Database	No
MongoDB	MongoDB	Database	No
Oracle	Oracle Database Server	Database	No
PostgreSQL Global Development Group	PostgreSQL	Database	Yes
Oracle	MySQL	Database	Yes
Adobe	Adobe Acrobat	Office Tools	No
Microsoft	Microsoft Office	Office Tools	No
Apache Software Foundation	Apache Openoffice	Office Tools	Yes
The Document Foundation	LibreOffice	Office Tools	Yes
Apple	MacOS	Operating System	No
Microsoft	Windows 10	Operating System	No
The Linux Foundation	Linux Kernel	Operating System	Yes
Google	Android	Operating System	Yes
Canonical	Ubuntu Linux	Operating System	Yes
Parallels	Parallels	Virtual Machine	No
Vmware	VMware Fusion	Virtual Machine	No
Citrix	Citrix Hypervisor	Virtual Machine	Yes
Oracle	Oracle VM Virtualbox	Virtual Machine	Yes
Red Hat	Red Hat Virtualization	Virtual Machine	Yes
Microsoft	Edge	Web Browser	No
Google	Google Chrome	Web Browser	No
Apple	Safari	Web Browser	No
Mozilla Foundation	Mozilla Firefox	Web Browser	Yes

Specific products in each category were selected on the basis of having >40 CVE database entries and being actively maintained as of 2023. This list is hardly comprehensive of all software meeting these conditions. To diversify the social circumstances behind each product, preference was given to products made by companies that were not already included in the same category (e.g. iOS was not listed under Operating Systems because MacOS, also developed by Apple, was chosen instead). Likewise, in an effort to avoid counting the same vulnerabilities multiple times, products that were not derived from or dependent on the same source code were preferentially included when possible (e.g. only two web browsers based on the highly popular Chromium framework, Chrome and Edge, were included, despite others potentially meeting the basic inclusion requirements).

For each product, I took the average CVSS score of all associated entries in the CVE database as an approximate measure of vulnerability severity. I then gathered the average ratings from and salaries of software engineers at the companies that develop these products on the employer review website Glassdoor to quantify how much value each assigns to engineers, under the assumption that receiving higher pay and expressing higher job satisfaction is indicative of being more valued. Next, I examined relevant financial documents (those being primarily SEC filings) from these companies to determine the pay, or perceived value, of executive staff. And finally, I gathered CVE data from a sample of open-source projects within the same categories.

Drawing on SCOT's concept of technological constructivism, which suggests that organizational dynamics between stakeholders alter the course of a given technology's development, the primary focus of my analysis will be on determining the nature and degree of correlation between the stakeholder power metrics and the vulnerability metrics. SCOT also

motivates my analysis of the differences in vulnerability severity between open-source and proprietary software projects, which tend to involve very different social structures.

Results

I gathered a total of 30,891 CVE (17,800 open-source, 13,091 proprietary) reports across 24 products (11 open-source, 13 proprietary) produced by 17 distinct organizations. In the data I collected, there did not appear to be any strong correlation between the average CVSS score of software products and the compensation or satisfaction of executives or software engineers (SWEs), nor the ratio of the two compensation statistics, at the responsible company. This result is interesting, as it suggests that pay and satisfaction may not be accurate measures of how stakeholders are valued within a corporate hierarchy, or at least of their ability to influence the direction of a project. Regarding organization type, open-source software projects tended to have notably lower CVSS scores than proprietary software on average, with CVSS scores on open-source projects averaging 7.29 as opposed to 7.84 across proprietary projects. A mean CVSS difference of ≥0.50 was observed within all comparisons between software of the same type, apart from web browsers, for which open-source and proprietary products were nearly equal.

Before discussing each relationship in further detail, a summary of the data is as follows:

Table 2. Summary of research data gathered (Source: Diamond, 2024).

Variable Name	Mean Value	Standard Deviation	Minimum Value	Maximum Value
Mean CVSS Score	7.52	0.71	5.40	8.80
Med. SWE Salary	$200,230.77	$42,093.46	$122,000.00	$266,000.00
Mean SWE Review	4.27	0.17	3.90	4.50
Mean Exec. Comp.	$22,045,803.55	$17,207,584.72	$4,938,815.07	$69,810,462.60

The first four relationships I analyzed dealt only with proprietary software, as open-source projects lack comparable statistics due to their organizational differences.

Figure 1. Plotting and Linear Regression of SWE Salary (upper left), Executive Compensation (upper right), Ratio of Executive to SWE pay (lower left), and SWE Glassdoor Reviews (lower right) on CVSS Scores (Source: Diamond 2024)

In the case of SWE salary, executive compensation, and their ratio, the lack meaningful linear correlation with vulnerability severity is clearly seen in Figure 1. While the sample sizes are small and cover a variety of product types, the vast range of compensations in my dataset, as seen in the data summary above, makes it more likely for any relationships to emerge, but none are visually apparent in the plotted data.

The Glassdoor review data suggests a slightly stronger relationship between vulnerability severity and job satisfaction, where more satisfied SWEs are likely to work on more vulnerable code. However, there was far less variation than expected in the satisfaction metric, with the difference between the minimum and maximum average review scores being 0.6 on a five-point scale, so this finding may be less likely to hold for larger samples with greater ranges.

The concept of interpretive flexibility from SCOT suggests that, due to differing values between the two groups, projects influenced more by executives than engineers would develop with different qualities than projects for which the opposite is true. While salary (and to some extent, job satisfaction) is often thought to be a measure of how much a company values a group of employees, the lack of any apparent relationship between this statistic and vulnerability severity suggests that either pay was a poor measure of value in these cases, or that on average, highly-valued stakeholders may not have as much influence over a software project's security as strongly as one would expect. Another alternative explanation for this result would be that, in the sampled products, engineers and executives valued security to approximately the same degree.

A far stronger correlation was observed between vulnerability severity and organizational structure, where open-source products appeared to have a lower average CVSS score than proprietary projects of the same type across a large sample of CVE reports. The sole exception to this clear trend is web browsers, though there is no readily apparent social explanation as to why. Web browsers are arguably the most complex and rapidly-evolving type of software, so technical factors may be responsible.

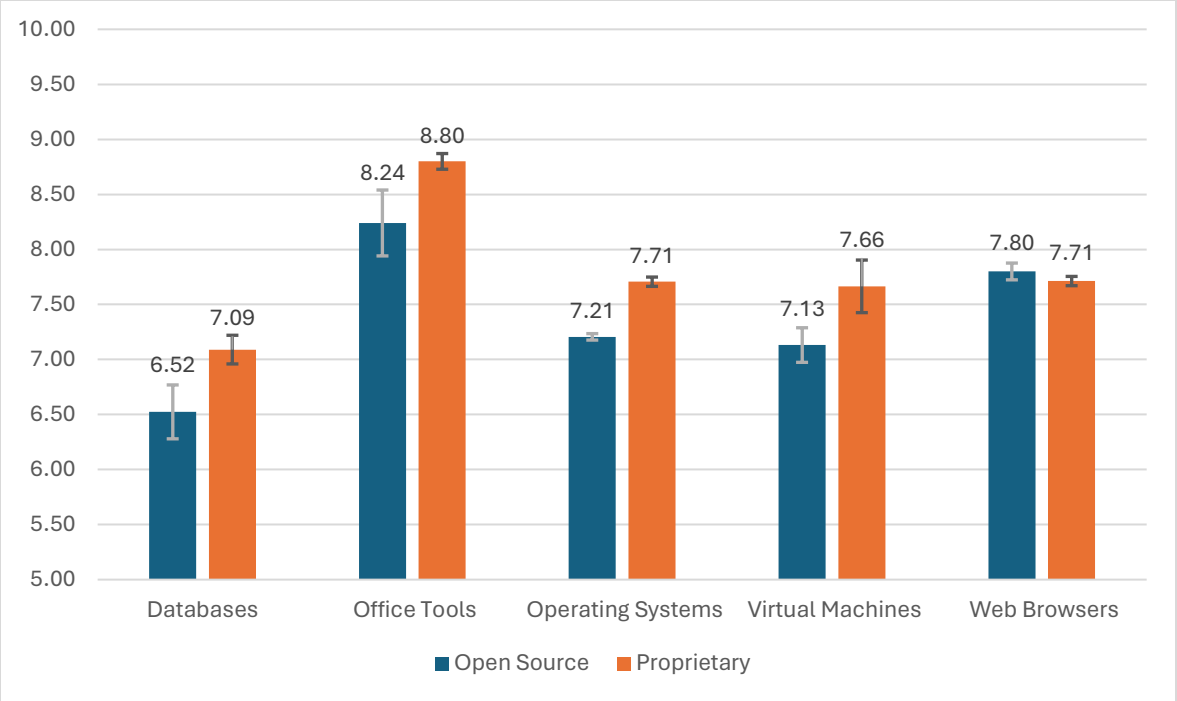

Figure 2. Weighted Mean CVSS Scores of Software Products by Type, with 95% Confidence Interval Standard Error Bars (Source: Diamond 2024)

SCOT suggests that this variance is impacted by the greatly different social structures between companies and open-source communities,. The lack of interaction between managerial and engineer shareholders is among the starkest social contrasts between open-source and proprietary projects, and as such, could be a contributor towards these notable differences in security. It may also be that the dynamics between engineers themselves is a factor. In companies, engineers tend to interact directly in teams on a daily basis, while open-source communities are less centralized and more individual-oriented, with developers entering and leaving the communities freely. There is also the lack of direct financial interests in open-source software, which draws in different developers than proprietary software – many individuals working on proprietary software at companies likely would not continue to do so for free, even if

their workplace were to undergo organizational change, while the same obviously cannot be said for open-source developers, who are already volunteers. The answer is likely some combination of these factors.

Discussion

My analysis of how organizational structure relates to software vulnerability severity is one example of an attempt at studying the social and otherwise nontechnical contributors to software quality. Though nowhere near as much research has been done on nontechnical factors as technical factors in software quality, other works have focused more on individual-level social factors, such as how software engineers sense of control and pleasure with their work positively impacts the quality of their code (Lundestad & Hommels, 2006), or how communication skills have a weaker impact on quality (Maria Jose Salamea & Carles Farre, 2019).

There are some limitations inherent to studying proprietary software. The lack of published vulnerability data leads to reliance on the imperfect CVE scoring system, as was acknowledged upon its introduction in this paper. Likewise, employers do not tend to publish salary or job satisfaction data, leaving crowdsourcing websites like Glassdoor as a publicly accessible approximation for developer salaries across the company as a whole, rather than by specific team or project. And while Glassdoor's estimates of job satisfaction tend to be close to other measurements (Landers et al., 2019), there is less empirical evidence that Glassdoor salaries always represent the real, modern salaries of workers in the relevant positions. On the subject of "modern salaries," the fact that Glassdoor does not allow queries for historical salary or rating data also limited the scope of research done with their platform, including this paper. A final, very significant limitation is survivorship bias. All the products I selected for this research

were successful enough to have vulnerability analysts care to document problems with them. There have been plenty of database systems, web browsers, office tools, and other products that have not reached this level of influence, but low-adoption or failed projects are inherently difficult to gather data on, limiting their ability to be used as samples.

In my future engineering practice, I will give more consideration to the social structure of projects I work on, and try to determine how any groups I collaborate with could best organizationally benefit all internal and external stakeholders. More broadly, I will pay attention to social contributors to software quality, and encourage others to do the same. While the immediate technical questions of "what line of code is the fault on?" and "should integration testing account for this case?" are understandably compelling and important for us to answer as engineers, my research suggests that we would benefit from supplementing these questions with more software quality-promoting social factors in the workplace. It is also clear that we need a better understanding of what these quality-promoting social factors are to begin with, both organizationally and among individual stakeholders. I would like to see social frameworks for software quality develop and become routinely implemented alongside existing technical frameworks like software testing, which already see wide use and study (Rafique & Misic, 2013).

Conclusion

There is still significant research to be performed in the area of this thesis. Future work could involve more specific analysis of how stakeholders interact within open-source and corporate environments, or broadening out to involve other types of organizations. In the former case, dealing with more direct interactions between stakeholders than pay ratios could show

interesting results. Some research has been conducted on software quality amid conflicts between different stakeholder groups in corporate environments (Wong, 2005), but there seems to be less material on inter-developer conflicts, which are particularly relevant to open-source projects. Studies also could be done on cooperation between stakeholders, or frequency/degree of stakeholder interaction on a project, and how these factors influence software quality. In the case of expanding to more organizational structures, there are a few groups to choose from. Non-profit organizations tend to have somewhat similar structures to for-profit corporations, but without as much of a monetary incentive for the involved parties, making them a potentially interesting middle ground comparison. Software can also be developed by groups of friends, academic or research organizations, governmental agencies, and in many other vastly different social structures. And that is not even to begin exploring variations on these structures, like remote-work-only software companies, which are becoming increasingly prevalent. There is significant room for exploring the differences between these groups in terms of software quality.

Regardless of the future direction of social research on software quality, I hope that this paper has shown that there are important relationships to be found between the technical and social aspects of software development. With how much we currently rely on the flawless operation of various software products in our everyday lives, any factor that influences the quality of code matters. And by only focusing on the technical aspects of quality, we may be rendering ourselves vulnerable to future socially-driven software disasters. Ultimately, as engineers, we must do our best to refine every aspect of our practice, and remember that we can never divorce the quality of software from the inherently social process of its development.

References

Alshahwan, N., Harman, M., & Marginean, A. (2023). Software Testing Research Challenges: An Industrial Perspective. *2023 IEEE Conference on Software Testing, Verification and Validation (ICST)*. https://doi.org/10.1109/icst57152.2023.00008

Barr, E. T., Harman, M., McMinn, P., Shahbaz, M., & Yoo, S. (2015). The Oracle Problem in Software Testing: A Survey. *IEEE Transactions on Software Engineering, 41*(5), 507–525. https://doi.org/10.1109/tse.2014.2372785

Bergman, P. (2018, July 17). *Boeing Forecasts $15 Trillion Commercial Airplanes and Services Market - Jul 17, 2018*. MediaRoom. https://boeing.mediaroom.com/2018-07-17-Boeing-Forecasts-15-Trillion-Commercial-Airplanes-and-Services-Market

CVSS v4.0 Specification Document. (n.d.). FIRST — Forum of Incident Response and Security Teams. https://www.first.org/cvss/specification-document#:~:text=The%20Common%20Vulnerability%20Scoring%20System

Ethiopian Civil Aviation Authority. (2022). Investigation report on accident to the B737-MAX8 Reg. ET-AVJ operated by Ethiopian Airlines. In *Bureau of Enquiry and Analysis for Civil Aviation Safety*. https://bea.aero/fileadmin/user_upload/ET_302__B737-8MAX_ACCIDENT_FINAL_REPORT.pdf

Florea, R., Stray, V., & Sjoberg, D. (2023). On the roles of software testers: An exploratory study. *Journal of Systems and Software, 204*, 111742–111742. https://doi.org/10.1016/j.jss.2023.111742

Friedman, M. (1970, September 13). The social responsibility of business is to increase its profits. *The New York Times*. https://www.nytimes.com/1970/09/13/archives/a-friedman-doctrine-the-social-responsibility-of-business-is-to.html

Gotterbarn, D., Miller, K., & Rogerson, S. (1997). Software engineering code of ethics. *Communications of the ACM*, *40*(11), 110–118. https://doi.org/10.1145/265684.265699

Hooda, I., & Singh Chhillar, R. (2015). Software Test Process, Testing Types and Techniques. *International Journal of Computer Applications*, *111*(13), 10–14. https://doi.org/10.5120/19597-1433

Hosseini, M., Rasmussen, L. M., & Resnik, D. B. (2023). Using AI to write scholarly publications. *Accountability in Research*, 1–9. https://doi.org/10.1080/08989621.2023.2168535

Howland, H. (2021). CVSS: Ubiquitous and Broken. Digital Threats: Research and Practice, 4. https://doi.org/10.1145/3491263

Karl, K. A., Peluchette, J. V., & Aghakhani, N. (2021). Virtual Work Meetings During the COVID-19 Pandemic: The Good, Bad, and Ugly. *Small Group Research*, *53*(3), 104649642110152. https://doi.org/10.1177/10464964211015286

Katalon, Cigniti, & Deloitte. (2023). *State of Software Quality 2023*. Katalon. https://katalon.info/hubfs/download-content/report/2023/state-of-software-quality-2023.pdf

Kenton, W. (2021, August 13). *What Is Real Time?* Investopedia. https://www.investopedia.com/terms/r/real_time.asp

Krasner, H. (2022). The Cost of Poor Software Quality in the US: A 2022 Report. In *CISQ* (p. 3). Consortium for Information & Software Quality. https://www.it-cisq.org/wp-content/uploads/sites/6/2022/11/CPSQ-Report-Nov-22-2.pdf

Kroeger, T. A., Davidson, N. J., & Cook, S. C. (2014). Understanding the characteristics of quality for software engineering processes: A Grounded Theory investigation.

Information and Software Technology, *56*(2), 252–271. https://doi.org/10.1016/j.infsof.2013.10.003

Landers, R., Brusso, R., & Auer, E. (2019). Crowdsourcing Job Satisfaction Data: Examining the Construct Validity of Glassdoor.com Ratings. *Personnel Assessment and Decisions*, *5*(3). https://doi.org/10.25035/pad.2019.03.006

Lundestad, C. V., & Hommels, A. (2006). Software vulnerability due to practical drift. *Ethics and Information Technology*, *9*(2), 89–100. https://doi.org/10.1007/s10676-006-9123-1

Maria Jose Salamea, & Carles Farre. (2019). Influence of Developer Factors on Code Quality: A Data Study. *UPCommons Institutional Repository (Universitat Politècnica de Catalunya)*. https://doi.org/10.1109/qrs-c.2019.00035

Martin, M. (2021). *Computer and internet use in the United States: 2018 American Community Survey Reports*. https://www.census.gov/content/dam/Census/library/publications/2021/acs/acs-49.pdf

Pinch, T. J., & Bijker, W. E. (1984). The Social Construction of Facts and Artefacts: Or How the Sociology of Science and the Sociology of Technology Might Benefit Each Other. *Social Studies of Science*, *14*(3), 399–441. http://www.jstor.org/stable/285355

Priestly, T. (2020). *Tech Troubles: The true cost of outdated workplace tech*. Techtalk.currys.co.uk; Currys, AMD. https://web.archive.org/web/20210210093723/https://techtalk.currys.co.uk/computing/workplace-productivity/

Rafique, Y., & Misic, V. B. (2013). The Effects of Test-Driven Development on External Quality and Productivity: A Meta-Analysis. *IEEE Transactions on Software Engineering*, *39*(6), 835–856. https://doi.org/10.1109/tse.2012.28

Ramler, R., & Wolfmaier, K. (2006). Proceedings of the 2006 international workshop on Automation of software test. *AST '06: Proceedings of the 2006 International Workshop on Automation of Software Test*. https://doi.org/10.1145/1138929

Robison, P. (2019, June 29). *Boeing's 737 Max Software Outsourced to $9-an-Hour Engineers*. Bloomberg.com. https://www.bloomberg.com/news/articles/2019-06-28/boeing-s-737-max-software-outsourced-to-9-an-hour-engineers

Sabater, A., & Asif, M. H. (2023, November 9). Private equity investments in systems software head for 5-year low. Www.spglobal.com. https://www.spglobal.com/marketintelligence/en/news-insights/latest-news-headlines/private-equity-investments-in-systems-software-head-for-5-year-low-78274257

Salahirad, A., Gay, G., & Mohammadi, E. (2023). Mapping the structure and evolution of software testing research over the past three decades. *Journal of Systems and Software, 195*, 111518. https://doi.org/10.1016/j.jss.2022.111518

Starks, R. (2023, May 3). Unity software is 1 stock to avoid in a bear market. Nasdaq. https://www.nasdaq.com/articles/unity-software-is-1-stock-to-avoid-in-a-bear-market

Topelson, D., Kira, R., Christopher, H., & Bavitz, T. (2017). *Governance for Open Source Projects*. https://clinic.cyber.harvard.edu/wp-content/uploads/2017/03/2017-03_governance-FINAL.pdf

Wong, B. (2005). Understanding Stakeholder Values as a Means of Dealing with Stakeholder Conflicts. *Software Quality Journal*, *13*(4), 429–445. https://doi.org/10.1007/s11219-005-4254-x

Wunder, J., Kurtz, A., Eichenmüller, C., Gassmann, F., & Benenson, Z. (2023, August 29). *Shedding Light on CVSS Scoring Inconsistencies: A User-Centric Study on Evaluating*

Widespread Security Vulnerabilities. ArXiv.org. https://doi.org/10.48550/arXiv.2308.15259

Zhang, S., Caragea, D., & Ou, X. (2011). An Empirical Study on Using the National Vulnerability Database to Predict Software Vulnerabilities. *Lecture Notes in Computer Science*, 217–231. https://doi.org/10.1007/978-3-642-23088-2_15