Javen W. Zamojcin
CSE870 Homework 1
2025/01/22

Of the myriad parties involved in the Therac-25 incidents [3] culpable to some extent, both proactively and reactively, the onus belongs most to the AECL Quality Assurance (QA) Manager. On the proactive side of any software system, the QA team serves as the final barrier between the product and the public; any bugs, design flaws, inconveniences, etc., are ultimately owned by the QA management. Evidently, as made abundantly clear in the FDA investigation reports [3, pp. 38-44], the AECL QA Manager responsible for the safety and testing of the Therac-25, was negligent in many respects. The reuse of faulty code [3, pp. 3, 48], the lack of evidence for any methodological code testing or error identification [3, pp. 40, 47, 48], repeatedly hasty failed solution attempts to known issues [3, pp. 45, 46], a lack of code audit trails [3, p. 40], cryptic malfunction messages [3, p. 42], the conflation of reliability with safety [3, p. 48], a failure to identify root error causes [3, pp. 41, 45], unnecessarily complex software design [3, p. 48], are just some of the examples of failures the AECL QA Manager was responsible for.

From my personal experience of working for a robotics company as a software deployment engineer, partly responsible for the safety and testing of dangerous robotics systems, the biggest mistake of the Therac-25 in my opinion, was the removal of the independent hardware safety system redundancies [3, p. 38]. Robotics systems by their nature are overwhelmingly complex, and quite literally can have a million moving concurrent components, both in software and hardware. I cannot enumerate the number of instances where the robots behaved unexpectedly due to some obscure software bug (which might take months to properly diagnose!), only to be caught by hardware safety redundancies. And we had many very talented engineers working around the clock on these software systems, so to move all safety functionality to the software side of the Therac-25 [3, p. 3] seems like pure hubris to me.

A number of similarities exist between the Therac-25 and Toyota “Unintended Acceleration” [5] incidents, but the overarching theme seems to be the hasty transition of traditionally mechanical systems to low-quality cumulative software systems, followed by a prolonged cover-up from the business end [5, para. 6]. Both the Therac-25 and Toyota throttle control software systems had many overly complex, unmaintainable, and untestable routines [3, p. 48] [5, para. 9]; both had inadequate failsafe systems and the general lacking of a safety architecture despite being critical-risk products [3, p. 47] [5, para. 9]; both allowed for single-point failures, or the removal of hardware safety redundancies [3, p. 3] [5, para. 15]; both failed to consistently follow industry-standard or in-house programming rules [3, p. 47] [5, para. 13], but with some leniency given here to the Therac-25 team due to the nascency of software engineering at the time; both were lacking a code peer-review system [3, p. 40] [5, para. 19]; and both had a general lack of transparency with their code and methodologies [3, pp. 22, 31] [5, para. 31]. Perhaps naively, a solution to help prevent future incidents like these is to require open-source code policy for critical-risk systems. There was no excuse for the Toyota accidents, given its recency and how mature the software engineering industry is at this point; early public accountability may have prevented their negligence.

In the relation of LLMs to these accidents, these tools potentially could have made the situation worse through the identification failure and further propagation of hidden race condition bugs [3, p. 35]. It is established that software-intensive systems generated by LLMs are often posed with various security weaknesses and other hidden bugs [1, p. 1]; and in general tend to struggle to generate code successfully for complex problems like the Therac-25, often producing shorter yet more complicated code solutions [2, p. 1]. Therefore, an over-reliance on LLM tools for the Therac systems could have resulted in even more cumulative hidden bugs and “spaghetti code”. However, on the other side of the argument, certain LLMs have actually been shown to identify and resolve complex multi-threading bugs whereas traditional static

analyzer tools failed [4]. Perhaps if these LLM tools were used to analyze the Therac-25 codebase early enough, downstream issues and deaths could have been prevented.

Bibliography

[1] A. Mohsin et al., “Can We Trust Large Language Models Generated Code? A Framework for In-Context Learning, Security Patterns, and Code Evaluations Across Diverse LLM,” arxiv.org, https://arxiv.org/pdf/2406.12513v1 (accessed Jan. 22, 2025).

[2] S. Dou et al., “What’s wrong with your code generated by large language models? an extensive study,” arXiv.org, https://arxiv.org/abs/2407.06153 (accessed Jan. 22, 2025).

[3] N. Leveson, “Medical Devices: The Therac-25,” Safeware: System Safety and Computers, http://sunnyday.mit.edu/papers/therac.pdf (accessed Jan. 22, 2025).

[4] M. A. Ferrag, “Analyzing code with AI: How LLM can identify a vulnerability missed by formal verification and...,” Medium, https://medium.com/@medaminefrg/analyzing-code-with-ai-how-llm-can-identify-a-vulnerability-missed-by-formal-verification-and-daa8d0f6e1d0 (accessed Jan. 22, 2025).

[5] “Toyota unintended acceleration and the Big Bowl of ‘Spaghetti’ code,” Safety Research & Strategies, Inc., https://www.safetyresearch.net/toyota-unintended-acceleration-and-the-big-bowl-of-spaghetti-code/ (accessed Jan. 22, 2025).