Comparison of LLMs In Mitigating Risks in Software Development

By Zorawar Jaiswal

Abstract

This paper explores how LLMs can act as automated risk-spotters throughout Software Development Life Cycle (SDLC). Four state-of-the-art models—ChatGPT o3, Qwen 3, Gemini 2.0 Flash, and Grok3 were fed real project artefacts such as a Gantt chart, architecture document, and code. Each model had to output risks in JSON using an ISO 31000 severity matrix. Results show that Grok 3 and Qwen 3 flagged the largest share of high-severity issues, but none of the models fully obeyed the strict prompt without retries. The study highlights both the promise and current limits of LLM-assisted risk management.

1. Introduction

A single mismanaged deployed script cost Knight Capital's group $440 million within a span of one hour (Eha, 2012). Netflix streamed the most expensive boxing fight and crashed out for most viewers (Yoon, 2024). These are a few examples from the pool of software mishaps that occur when software teams omit disciplined life-cycle processes.

Cerpa & Verner (2009) state that most of the software projects in 2007 failed due to underestimation, unrealistic client expectations, and risks not incorporated during the planning phase. Not much has changed over the years besides the numbers.

1.1 Software Development Life Cycle

Software Development Life Cycle (SDLC) is a way of developing a software product (Lekh & Pooja, 2015). The SDLC houses various phases critical to creating the end product, i.e., the software. The SDLC comprises (i) Planning, (ii) Requirements, (iii) Design, (iv) Implementation, (v) Testing, (vi) Deployment, and (vii) Maintenance.

The Planning stage concerns high-level goals. Additionally, it involves resource allocation and risk assessment (Davis et al., 1988). The Requirement Analysis undergoes a formal requirement gathering phase and freezes the scope for production (Nurmuliani et al., 2004).

The Design phase is often cited as the most critical phase of the SDLC. It incorporates various tasks such as creating the software architecture, defining the technologies to use, and patterns to implement. A wrong design may lead to high rework in the future (Shafiq et al., 2021). A good design scales the software as its usage and features increase, while a bad design incorporates patches.

The next phase is developing the actual software. A perfect design and appropriate resource allocation reduce code complexity and technical debt. A code review sub-phase ensures the code correlates to the design and architecture guidelines.

Testing remains vital for client acceptance and error handling. Test cases are the final validation check to assess whether the developed software conforms to the finalized requirements. Various types of testing include, but are not limited to, unit, integration, regression, black-box, and user acceptance testing.

The Deployment phase comprises processes relating to installing the software and providing it to the end users, and the documentation.

The final stage is Maintenance. It comprises providing bug fixes, optimizations, and security updates. It can be seen as a cycle where the requirements are again gathered (expected behaviour in case of a bug), the development takes place (fixing of the bug), the bug fix is tested, and finally deployed, making this into a cycle, thus the name Software Development Life Cycle.

1.2 Importance of SDLC

Modern programming languages are fluid to the extent that syntactically correct code may work but assures zero maintainability, scalability, or security. Whether that code conforms to certain standards is something hard to establish without standardization. SDLC as a framework structurally segregates the various phases of software development. This ensures that the development doesn't turn into untraceable decisions or technical debt. Lack of a framework, such as SDLC, would lead to unaccountable decisions, a mismatch of product and requirements, and fragile applications (Cerpa & Verner, 2009).

1.3 Scope of this paper

This paper explores the idea of integrating Large Language Models (LLMs) as tools to estimate and highlight potential risks throughout each phase of the SDLC. The concept remains that good software is well written. A well-written software is well-designed, and a well-designed software has clear requirements. LLMs as a tool can help identify the risks in various software artefacts created during various SDLC phases, such as a Gantt chart, a user story, and code.

This paper further explores the versatility of LLMs in identifying the type of risks. To draw a comparison, 4 major LLMs are experimented upon in a single project – The Recipe Generator. The 4 LLMs are (i) ChatGPT o3, (ii) Qwen 3, (iii) Gemini Flash 2.0, and (iv) Grok 3.

The important questions answered through this paper are (i) How well are LLMs able to identify the risks in a software project throughout the Software Development Life Cycle, (ii) What patterns are LLMs able to draw from the artefacts, and finally (iii) which LLM is the most appropriate tool that should be incorporated.

2. Literature Review

2.1 SDLC Models

SDLC has various models that are appropriate to different types of requirements. A few of those include Waterfall, Agile, and DevOps.

2.1.1 Waterfall

The waterfall model is the most fundamental model within the SDLC framework. It builds up sequentially. Each phase is dependent on its previous phase. The phases include (i) System Engineering, (ii) Requirement Analysis, (iii) Software Design, (iv) Coding, (v) Testing, (vi) Maintenance. The flow is in one direction, meaning no overlap or jumping between phases (Mishra & Mohanty, 2011). The waterfall model is highly suitable for strict requirements such as patient case reporting to medical agencies (strict GDPR and HIPPA rules) and is not preferable for evolving products, such as a restaurant directory company pivoting to food delivery. The reason for this is freezing of requirements before implementation.

2.1.2 The V Model

The V model builds on the waterfall and draws parallels (mimicking a V shape). The primary focus is on quality assurance, where tests correlate to specifications and debugging conforms to the design of modules (Mishra & Mohanty, 2011).

2.1.3 Spiral Model

The Spiral Model is iterative. Software development takes place over multiple cycles where each cycle comprises (i) Planning, (ii) Risk Analysis, (iii) Engineering, and (iv) Validation. The first prototype is analyzed, and its feedback forms the basis of planning for the next iteration, looping into a spiral shape by the third cycle (Mishra & Mohanty, 2011). This model focuses heavily on reusability but is not the best in terms of resource consumption.

2.1.4 Agile & DevOps Model

Layers of managers and overwhelming documentation backlog gave rise to agile. Agile as a model incorporates following certain values and principles from the agile manifesto which in summary are inclined towards focusing on development over documentation (Beck et al., 2001). Small teams of developers decide on tasks and deliver projects with sprints. The first few sprints focus on major features (deal breakers), the later ones focus on minute features and optimizations, making teams effective for shorter periods but expensive for longer projects. Agile suits best in case of evolving requirements (Cline, 2015).

DevOps extends Agile, automates build-test-release pipelines. Studies report 26× faster time to market and 7× fewer failures, enabling continuous risk mitigation in delivery (Cline, 2015). DevOps allows teams to implement features such as one-click rollbacks and deployments and version controlling of releases for efficient feature and bug tracking (Humble & Farley, 2010).

Table 1 – Major Milestones in SDLC

SDLC Phase	Milestone	Tool/Artefact
Planning	Approval of project charter (budget, scope, main risks)	Confluence or Google Docs
Requirements	Requirement Freeze	Epic selection in Kanban board.
Design	Architecture approval	UML / DFD diagrams
Implementation	Code Freeze	Package freeze in Git
Testing	Test report pass	Client acceptance/ QA lead approval
Deployment	Release passed	Github Actions, IG, IQ approval
Maintenance	Green SLA	Incident report in Jira/ServiceNow

Table 1 describes the major milestones in SDLC. In our Recipe Generator pilot, the generic SDLC milestones are simpler and more manual, such as story freezing on Monday.com, deployment on localhost.

2.2 Industry Survey on SDLC

The adoption of practices such as agile has been famous across the tech industry. Salesforce.com as a company transitioned into agile workflows to leverage incremental and impactful builds for force.com (Singhto & Phakdee, 2016).

Tata Consultancy Services (TCS) has developed a GenAI SDLC assistant – a product for businesses to develop code faster. They partnered with Google Cloud and Gemini create agents that assist throughout the SDLC process (Tata Consultancy Services, 2025).

Amazon launched Amazon Q – A Gen AI powered assistant to accelerate problem-solving and streamline tasks. Amazon Q can write user stories, set up Jira Boards, help develop, test, and debug code, leading to automation of most manual processes in SDLC (Makvana et al., 2024).

IBM partnered with Amazon for AI-powered SDLC solutions and claims to achieve 30% shorter development cycles, and a 60% decrease in time spent on analysis of requirements (Rivadulla & Colombatto, 2025).

2.3 Industry Survey on Software Risk Management

In 2024, Netflix streamed the biggest boxing fight starring Jake Paul and Mike Tyson. The hype was created, but the infrastructure was not. The match led to a service outage for most of North American viewers. The non-functional requirements were not implicitly accounted for in this scenario (Yoon, 2024).

Risky software has previously led to huge financial losses, as seen in the case of the Y2K incident. The Y2K incident was assumed to affect the entire airline and banking sectors with the transition of date to a different century (1900s to 2000s). The systems were expected to crash since two-digit dates (year 1998 referred as 98) would have turned to 00 (Smithsonian, 2000).

Sensor failure in Ethiopian Airlines aircraft in 2019 sent noisy data to the control system, undesirably activating the maneuvering system, leading to the loss of life of nearly 190 people (Presse, 2022).

These trends strengthen the focus on software risk management across all sectors, developing software both internally and for customers.

2.4 Academia Survey on Software Risk Management

Pilliang et al. (2022) on the Stackoverflow 60K dataset from Kaggle by experimented sentiment analysis and K Means Clustering. The SO60K dataset is a collection of user posts. They analyzed sentiments of user questions, ranked the sentiment analysis, and estimated the likelihood of occurrence. They computed a risk matrix comprising the severity and occurrence.

Khurana & Wassay (2023) focus on the security risks during SDLC. They argue that security is the main driver of risks in software. The inference is that security risks may be invoked due to fast-evolving technology, while cybersecurity guidelines take time to improve. They propose having multiple standup calls to review updates as a safeguard.

2.5 Challenges and Emerging Trends

Scope creep. Agile backlogs make evolving requirements cheap only in theory. Any scope updates at the last minute without proper planning may lead to budget issues, uncontrolled growth and lead to reduced product quality (Mishra & Mohanty, 2011).

Technical debt. Rapid iteration often defers refactoring. Technical debt refers to using shortcuts to meet deadlines adding on to a pile of rework and higher regression costs. It leads to performance issues, a reduction in scalability, and more patchwork, costing money in the long run (Cerpa & Verner, 2009).

LLM hallucination and policy risk. LLMs can fabricate non-existent APIs or misclassify design choices as “critical” (Shafiq et al., 2021). Responsible-AI guidelines such as ISO/IEC 42001 now demand tracking, confidence scoring, and the right to explanation for automated decisions. They define how AI must be used in a project, irrespective of its application/domain. LLMs also cost plenty of resources to run locally and are not scalable on localhost.

3. Methodology

The 4 selected LLMs were input the same artefacts from the Recipe Generator Project. The first artefact was the software architecture document including the user story. The second artefact was the code and the last artefact was the Gantt Chart. The Recipe Generator project was developed using the Agile model, closely keeping SDLC in mind with the intent to generate incorrect and contradictory artefacts, making it a perfect fit for assessing LLM quality.

The LLMs were prompted to configure as an ISO 31000 certified software risk assessor and were asked to identify as many risks as they could and classify them. All artefacts, user prompts, and evaluation criteria are available in (Jaiswal, 2025). The next section discusses the results.

We ranked each risk with a 5 × 5 likelihood-by-impact matrix aligned to ISO 31000. Likelihood scores (1 = Rare up to 5 = Almost-Certain) and Impact classes (C1 = Negligible to C5 = Catastrophic) multiply to a severity score from 1 to 25. Scores map to labels: Info (1-3), Low (4-7), Medium (8-14), High (15-19), Critical (20-25).

The LLM was asked to categorize risk taxonomy tags for each identified risk between tags: REQ_NONFUNC, SECURITY, PLANNING, PERFORMANCE, QUALITY, DEPLOYMENT, MAINTAINABILITY, REDUNDANCY, and OTHER.

Cost metrics were logged per inference: LLM tokens used and latency of query resolution. A mix of risks was included in the artefacts, such as hardcoding keys, fixing paths, code not projecting on the user story, no test cases, and hardcoded scenarios.

4. Results and Discussion

Gemini performed the worst in the case of distribution by severity since it could not classify a single risk as high (even security vulnerabilities). Qwen had the most balanced approach out of all the LLMs with 55% risks classified as medium, 36% as High and 9% as low. Grok performed the second best in terms of versatility and chatgpt the thirds as it labeled most of the risks as medium.

ChatGPT performed the best in terms of finding the maximum vulnerabilities (16) followed by Grok (15), Qwen (11), and Gemini (9).

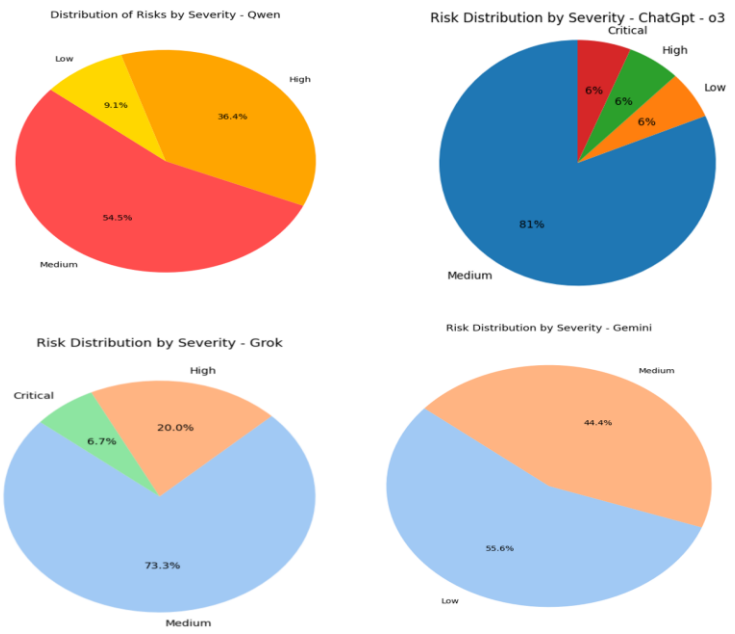

Fig 1. Distribution of Risks by Severity

Fig 2. Shows the risk type distribution per phase. ChatGPT was the chattiest out of all the LLMs and found of its risks from the software architecture document only. The points correlated such as – missing tests, no plans of deployment, but it repeated ‘unencrypted credentials thrice’ leading to

verbose results. Grok performed the best as it was the only LLM to highlight discrepancy between the code and the software architecture document. Qwen was the second best as it highlighted the most important security risk – prompt hacking via OS subprocess. Gemini performed the worst since it could not compete with its peers. Table 1 shows the scores for all of the LLMs.

Table 2 – Score of all LLMs

LLM	ChatGPT o3	Qwen 3	Grok 3	Gemini Flash 2.0
Distribution of Risks by Severity	Third	Best	Second	Worst
Number of Risks identified	16	11	15	9
Risk Type Distribution per phase	Best	Bad	Good	Worst
Most Critical artefacts found	3	4	3	1

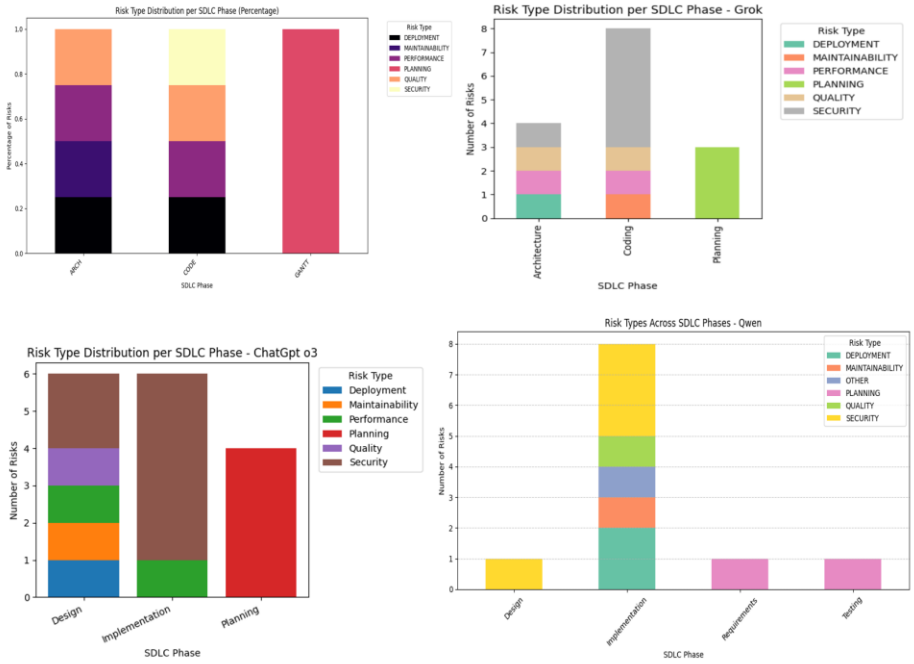

Fig 2. Risk Type Across SDLC Phases

None of the LLMs could highlight risks associated with the system's scalability. The application worked as a singleton object and was designed without scalability in mind. Additionally, only partial code was input to the LLMs, they were unable to correlate with the missing code.

5. Conclusion & Future Work

To recap, across 51 distinct risks, Grok 3 surfaced 5 Critical/High issues (29 %), Qwen 3 four (24 %), ChatGPT only two (12 %), and Gemini none for a single prompt. These findings suggest LLMs can identify SDLC risks but are bad at generalizing, with poor cross-artefact contradiction checks.

Out of all the LLMs tested, Grok was the most suitable in assessing and mitigating risks in a software project throughout all phases of SDLC. Qwen happened to be the second best. Even though ChatGPT is effective, it misses out on identifying and correlating missing patterns, and hallucinates to the extent of repeating stuff. Gemini happens to be the worst in the case of risk assessment, highlighting poor context awareness and a shallow thought process.

LLMs are a great tool to help enhance SDLC and identify risks, as most of the risks were identified on the first prompt via the Software Architecture Document, advocating for the statement – "A good design is crucial to good software".

The suggested future work for this domain would be experimenting with (i) multiple projects and internal communication, such as meeting notes. (ii) Emphasis on cost-benefit analysis and feasibility analysis could be done, (iii) the AI-generated risks could be validated with a human expert ground truth to understand the exact gap between AI and human risk analysis.

References

Beck, Grenning, Martin, Beedle, Bennekum, Cockburn, Cunningham, Highsmith, Hunt, Jeffries, Kern, Marick, Martin, Mellor, Schwaber, Sutherland, & Thomas. (2001). Manifesto for Agile Software Development. https://agilemanifesto.org/

Cerpa, N., & Verner, J. M. (2009). Why did your project fail? *Communications of the ACM, 52*(12), 130–134. https://doi.org/10.1145/1610252.1610286

Cline, A. (2015). *Agile development in the real world*. Apress.

Darandale, S., & Mehta, R. (2022). Risk assessment and management using machine learning approaches. *2022 International Conference on Applied Artificial Intelligence and Computing (ICAAIC)*, 663–667. https://doi.org/10.1109/icaaic53929.2022.9792870

Davis, A. M., Bersoff, E. H., & Comer, E. R. (1988). A strategy for comparing alternative software development life cycle models. *IEEE Transactions on Software Engineering*, *14*(10), 1453–1461. https://doi.org/10.1109/32.6190

Eha, B. P. (2012, August 9). *Is Knight's $440 million glitch the costliest computer bug ever?* CNN. https://money.cnn.com/2012/08/09/technology/knight-expensive-computer-bug/

Humble, J., & Farley, D. (2010). *Continuous delivery: Reliable software releases through build, test, and deployment automation, Video Enhanced Edition*. Addison-Wesley Professional.

Jaiswal, Z. (2025, April 30). *Artefacts*. Google Drive. https://drive.google.com/drive/folders/1doeG1cK8-Lf0aALVovx3I2lYkwdpGWrG

Khurana, S. K., & Wassay, M. A. (2023). Towards challenges faced in agile risk management practices. *2023 International Conference on Inventive Computation Technologies (ICICT)*, 937–942. https://doi.org/10.1109/icict57646.2023.10134188

Lekh, R., & Pooja. (2015). Exhaustive study of SDLC phases and their best praxctices to create CDP model for Process Improvement. *2015 International Conference on Advances in Computer Engineering and Applications*, 997–1003. https://doi.org/10.1109/icacea.2015.7164852

Makvana, C., Saxena, S., & Vasudevan, V. (2024, May 8). *Accelerate your software development lifecycle with Amazon Q | AWS Devops & Developer Productivity Blog*. Amazon Web Services. https://aws.amazon.com/blogs/devops/accelerate-your-software-development-lifecycle-with-amazon-q/

Mishra, J., & Mohanty, A. (2011). *Software Engineering*. Pearson Education, Inc.

Nurmuliani, N., Zowghi, D., & Powell, S. (2004). Analysis of requirements volatility during software development life cycle. *2004 Australian Software Engineering Conference. Proceedings.*, 28–37. https://doi.org/10.1109/aswec.2004.1290455

Pilliang, M., Munawar, Hadi, M. A., Firmansyah, G., & Tjahjono, B. (2022). Predicting risk matrix in software development projects using bert and K-means. *2022 9th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, 137–142. https://doi.org/10.23919/eecsi56542.2022.9946637

Presse, A.-A. F. (2022, December 23). *Inquiry into 2019 Ethiopian Air Crash Confirms Software Failure*. barrons. https://www.barrons.com/news/inquiry-into-2019-ethiopian-air-crash-confirms-software-failure-01671821708

Rivadulla, J. M. P., & Colombatto, D. (2025, January 14). *IBM + AWS: Transforming Software Development Lifecycle (SDLC) with Generative AI*. IBM. https://www.ibm.com/products/blog/ibm-aws-transforming-software-development-lifecycle-sdlc-with-generative-ai

Shafiq, S., Mashkoor, A., Mayr-Dorn, C., & Egyed, A. (2021). A literature review of using machine learning in software development life cycle stages. *IEEE Access*, *9*, 140896–140920. https://doi.org/10.1109/access.2021.3119746

Singhto, W., & Phakdee, N. (2016). Adopting a combination of scrum and waterfall methodologies in developing tailor-made SAAS products for Thai service and manufacturing smes. *2016 International Computer Science and Engineering Conference (ICSEC)*, 1–6. https://doi.org/10.1109/icsec.2016.7859882

Smithsonian. (2000). *Y2K*. National Museum of American History. https://americanhistory.si.edu/collections/object-groups/y2k

Tata Consultancy Services. (2025). *TCS SDLC assistant: A perfect companion for developers, Delivery Teams*. TCS SDLC Assistant: A Perfect Companion for Developers, Delivery Teams. https://www.tcs.com/what-we-do/services/cloud/google/solution/tcs-sdlc-assistant

Yoon, J. (2024, November 16). *Netflix livestream outages reported during Mike Tyson-Jake Paul Fight - the new york times*. NY Times. https://www.nytimes.com/2024/11/16/business/media/netflix-outage-crash-boxing.html