Using LLMs to Bridge the Gaps in QA Test Plans at Firefox

John Pangas, *University of Calgary, Calgary, Alberta, Canada*
Suhaib Mujahid, *Mozilla, Montréal, Québec, Canada*
Ahmad Abdellatif, *University of Calgary, Calgary, Alberta, Canada*
Marco Castelluccio, *Mozilla, London, United Kingdom*

Abstract—As software systems grow in scale and complexity, ensuring their reliability becomes increasingly challenging due to factors like diverse platforms, rapid release cycles, and evolving user needs. This places greater demands on software quality assurance (QA), where comprehensive test planning is crucial. However, in addition to this process being manual and time-consuming, skilled QA engineers may potentially overlook critical scenarios.
To bridge this gap, we used an LLM, GPT-4 Turbo, to generate test plans for eight Firefox features, evaluating them for novelty, validity, and relevance. Our results showed that 27% of the test cases in LLM-generated plans surfaced previously missed test scenarios the QA team deemed valuable, while 50.5% replicated existing ones. Although 22.5% were invalid and out of scope, our approach shows potential for improving test coverage. In this paper, we share our experience with this methodology and offer insights for SE/QA practitioners integrating LLMs into their workflows.

INTRODUCTION

Software Quality Assurance plays a crucial role in ensuring products adhere to quality standards through systematic testing and validation. This is particularly important as digital systems increasingly permeate business processes and consumer interactions across industries. A recent report shows that the cost incurred by U.S. companies due to poor software quality has risen to at least $2.41 trillion [1]. Therefore, the importance of comprehensive testing has become increasingly critical in today's competitive software landscape. Furthermore, as software systems grow more complex, these challenges become increasingly pronounced and impact software quality and release schedules.

A core artifact in the validation process is the QA test plan. These plans specify the functionality to be tested, the expected outcomes, and provide a blueprint for systematically evaluating software features. At Mozilla, every new feature is accompanied by a test plan containing high-level test cases that outline key functionalities and expected behavior under specific conditions. For example, a test case for a PDF reader feature might be: "*Verify that the PDF reader view works in the light theme*" For a single feature, engineers typically author between 10 to 30 such test cases.

The process of constructing test plans is intricate. QA engineers must analyze feature specifications, understand real-world user workflows, and anticipate potential edge cases and integration pitfalls. This is a manual, time-intensive task that relies heavily on the engineer's experience and domain familiarity. As a result, even seasoned professionals may inadvertently miss important test cases. This is especially true for scenarios involving rare or unexpected behaviors. Such omissions can lead to gaps in test coverage that may compromise the quality of the functionality of features and interfere with the user experience.

While recent work has explored automating aspects of software testing, particularly the generation of code-based unit tests and test cases [2], there has been, to the best of our knowledge, no focus on supporting QA engineers in creating test plans from natural language feature specifications.

Therefore, to address this gap and explore how the challenges of manual test plan creation can be mitigated, we leveraged a large language model (GPT-4 Turbo) to assist in generating test plans. Our focus was specifically on identifying potential test cases that may have been overlooked by QA teams at Mozilla,

XXXX-XXX © IEEE
Digital Object Identifier 10.1109/XXX.0000.0000000

and to that end, we applied our approach across eight Firefox features currently under development.

For each software feature under testing, we provided its description and test scope as input to the LLM, prompting it to generate a test plan with 25 test cases. To assess the effectiveness of these generated test plans, we collaborated with Mozilla QA engineers, who evaluated each of the test cases for novelty, validity, and relevance using a structured feedback form.

Based on their feedback, 77.5% of the test cases in the LLM-generated test plans were considered valid and relevant, with 27% introducing new test scenarios that human testers had previously overlooked. However, challenges remain, 14.5% of the test cases were invalid, and 8% were unsuitable due to contextual or scope-related issues. Throughout this paper, we share our experience by detailing our approach, key insights, and practical recommendations for SE/QA practitioners. We make our replication package publicly available to facilitate research in this area[1].

RELATED WORK

Large Language Models (LLMs) like Gemini, LLaMA, GPT series, and Mistral have been increasingly utilized in software engineering, particularly for code generation, program repair, and code comprehension. In software testing, LLMs have been applied to various stages of the testing lifecycle, unit test generation [3, 4], and test oracle creation [5, 6]. Earlier studies focused on fine-tuning LLMs with domain-specific datasets to enhance test generation quality, readability, and coverage [4, 7]. More recent research has shifted toward prompt engineering, using adaptive contexts to inject project-specific information into test generation tasks [8, 9].Beyond standalone test case creation, LLMs have also been combined with external documentation such as API references [10] and bug reports [11] to improve test relevance and effectiveness.

Most prior work has focused on generating executable unit test code [3, 4], typically expressed in programming languages and and targeting single functions or methods. In contrast, our approach produces test plans containing high-level, scenario-driven test cases written entirely in natural language and organized with preconditions, and expected outcomes rather than code. Creating such test plans involves reasoning about complete feature requirements and anticipating how end users engage with the system, rather than simply checking whether individual functions behave correctly. By doing so, our contribution complements prior research by operating at a different layer of the testing stack. Whereas unit tests establish correctness at the code level, our test plans emphasize validation of feature behavior from the user's perspective to support both manual and automated QA workflows in complex GUI-based systems such as the Firefox Browser.

To the best of our knowledge, no prior work has specifically addressed the use of LLMs for comprehensive software QA test plan generation. Our work takes an initial step in this direction, aiming to empower QA teams with more complete, structured, and human-readable test scenarios to enhance overall test coverage.

APPROACH

To evaluate how large language models can assist in high-level QA test planning, we designed an approach that prompts GPT-4 Turbo to generate test plans based on real feature details from Firefox. Our methodology involved prompt selection, temperature tuning, and prompting GPT-4 Turbo. We describe this approach in more detail in the following subsections.

Using GPT-4 Turbo

For our implementation, we chose GPT-4 Turbo, one of the most advanced LLMs available at the time and the latest version of ChatGPT. It has been widely applied in software engineering prior work [12, 13, 14].

Model Parameter Selection

To encourage creativity in generating test scenarios that the QA team might have overlooked, we experimented with different model temperature settings (i.e., 0.5, 0.8, and 1.0).We conducted preliminary experiments by prompting the LLM to generate test plans for current Firefox features and assessing the outputs in terms of diversity, relevance, and validity. At a temperature of 1.0, the outputs were more diverse but often irrelevant or invalid; at 0.5, the results were highly predictable and closely resembled existing test plans, offering little novelty. A setting of 0.8 provided the most effective balance between novelty and practical relevance. To obtain largely consistent and reproducible outputs, we set the `seed` parameter to a fixed value across all model runs, while also maintaining the same temperature and prompt.

Few-shot Strategy and Prompt Template

To enable the LLM to generate test plans, we adopted a few-shot prompting strategy. The few-shot examples

[1] https://zenodo.org/records/16944475

Table 1: Overview of the Eight Features.

Feature	Functionality
Select Translations	Allows translating highlighted text rather than the whole page.
Text Input Event	A prototype of the beforeinput event that supports legacy browser behavior.
macOS Session Resume Support	Restores Firefox after macOS restarts to improve session recovery.
Image to Text GenAI in PDF	Suggests alt text for images in PDFs using generative AI.
Back Button Intervention	Prevents users from getting trapped by unexpected navigation entries.
Reader View Comfort	Enhances reader view for better accessibility and ease of reading.
Text Fragments	Highlights a text snippet in the URL fragment for navigation.
PiP auto-trigger	Automatically enables Picture-in-Picture mode for active videos when switching tabs.

consisted of the test plans previously curated by the QA team for three stable Firefox features: *Reporting Tool*, *Highlight Text Button*, and *Tab Hover Preview*. There was no manual selection or filtering of the test cases from these test plans. We chose these features to guide the LLM in learning the structure, tone, and phrasing typical of Mozilla QA test plans. This ensured that QA engineers could use the outputs directly without additional modifications, while also providing the model with contextual exposure to related browser components, such as themes and layouts, which could support better reasoning about cross-feature interactions during test plan generation.

Figure 1 shows the prompt template used to generate the test plans. This template guided the LLM in understanding the task, context, and expected output.

Features for Evaluation

To assess the effectiveness of LLMs in generating test plans, we used eight new Firefox features that were under active development. Table 1 presents a general overview of the eight features and their functionalities. These features were accessible only to users of Firefox Nightly Build at the time of our evaluation. Importantly, the feature descriptions and test scopes were not publicly available, meaning they could not have been included in the pretraining data of LLMs like GPT-4 Turbo. This eliminates data leakage and ensures the model's responses were uninfluenced by prior exposure, making our evaluation a realistic test of its ability to generate novel test plans.

GPT-4 Turbo was prompted only once per feature to produce a test plan with 25 test cases for the QA engineers to evaluate. This single-execution approach avoided the time cost of repeatedly prompting the LLM during QA workflows, while also reducing token consumption and energy usage from reviewing multiple large sets of test plans, thereby improving the overall practical feasibility of the method.

FEEDBACK ON THE LLM GENERATED TEST PLANS

To evaluate our approach, we collaborated with three lead QA engineers at Mozilla, each of whom was a QA team lead over the specific Firefox features. For each feature under study, the corresponding lead QA engineer, who was responsible for its QA and familiar with its specifications and existing test plans, reviewed that feature's LLM-generated test plan. Using a structured feedback form, they evaluated each test case based on novelty, relevance, and validity, selecting from four predefined labels: "Yes: Novel" for additions that were new, valid, and relevant, "Already Exist" for test cases that were valid and relevant but similar to existing ones, "No: Invalid" for invalid test cases, and "No: Other" for those that were irrelevant or otherwise not fitting the intended testing purposes. To gain deeper insights into the effectiveness of our approach, we included three open-ended questions focused on what the QA engineers viewed as strengths and potential areas for improvement. Specifically, we asked why certain test cases were excluded under "No: Other", whether the generated test plans inspired new ideas, and if any test cases required modification. To reduce the potential for labeling bias, the first author also independently evaluated each feature's generated test cases for novelty, relevance, and validity. We found that the annotators achieved almost perfect agreement (kappa = 0.91)[15].

Table 2 presents the results of this evaluation and summarizes the distribution of test plan classifications across all the features.

Novelty and Impact

A key insight from the results is that 27% of the test cases in the LLM-generated test plans for all features were classified as "Yes: Novel", meaning that approximately one in four introduced new valid and relevant testing scenarios previously overlooked by QA engineers. For example, in the *PiP Auto-trigger* feature,

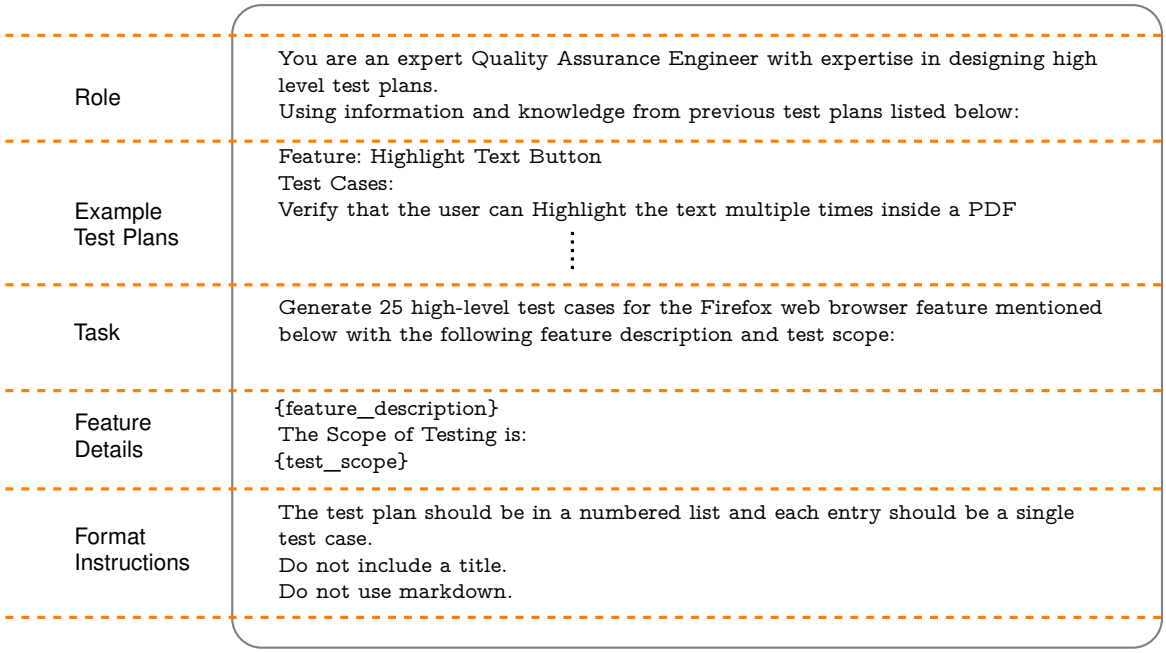

Figure 1: The Prompt Template with examples

Table 2: Acceptance and Rejection Rates of Test Cases in the Test Plans

Feature Name	Yes: Novel	Already Exist	No: Invalid	No: Other
Text Fragments	11 (44%)	13 (52%)	0	1 (4%)
Back Button Intervention	0	14 (56%)	6 (24%)	5 (20%)
Text Input Event	6 (24%)	4 (16%)	13 (52%)	2 (8%)
Image to GenAI in PDF	8 (32%)	11 (44%)	5 (20%)	1 (4%)
MacOS Session Resume Support	12 (48%)	9 (36%)	0	4 (16%)
Reader View Comfort	1 (4%)	19 (76%)	2 (8%)	3 (12%)
Select Translations	4 (16%)	20 (80%)	1 (4%)	0
PiP Auto-trigger	12 (48%)	11 (44%)	2 (8%)	0
Total	54 (27%)	101 (50.5%)	29 (14.5%)	16 (8%)

the LLM generated new test cases, such as verifying Picture-in-Picture functionality with videos requiring login or subscription access. Similarly, for the *Select Translations feature*, the LLM proposed an innovative test case to verify that the translation option does not appear on websites that hijack the context menu.

The degree of novelty varied across features. This variation might stem from differences in feature complexity and scope. Features with broader interaction possibilities, like *PiP Auto-trigger*, resulted in more novel test cases, possibly because the LLM could leverage its broad pre-trained knowledge to imagine diverse usage scenarios. Conversely, narrowly scoped features like *Reader View Comfort* showed higher rates of duplicate test cases, suggesting that the limited feature scope naturally constrains the possible unique test scenarios. Interestingly, even when the LLM failed to generate valid test cases, its outputs occasionally inspired new testing ideas. For example, while the *Back Button Intervention* feature did not receive any valid novel test cases, the QA team noted that an invalid test case suggested stress-testing history functionality, prompting them to consider applying this idea in a more valid testing scenario.

Capturing QA Thought Processes

A substantial 50.55% of test cases from all features were labeled “Already Exist”, indicating that the LLM successfully mirrored the patterns and thought processes of human testers. The degree of similarity varied by feature–*Reader View Comfort* had the highest overlap (76%), suggesting the LLM closely followed existing testing strategies, whereas *Text Input Event* had only 16% overlap, meaning it introduced a higher proportion of novel or irrelevant test cases. Based on the responses from the feedback form, the QA engineers reported that there was no need to modify the structure

or language of the generated test plans. In certain cases, some of the test cases from the LLM-generated test plans were even preferred over manually created ones. For example, in the *macOS Session Resume Support* feature, the QA team noted that the LLM provided better summaries, leading them to replace some of their own manually written test cases with the LLM-generated versions.

Limitations in Generated Test Plans

14.5% of the test cases were rejected as "Invalid", primarily due to misinterpretations of feature descriptions. For instance, in the *Text Input Event* feature, half of the rejected test cases were due to the LLM misunderstanding the word "before" in the *beforeinput* event. It incorrectly assumed that the *beforeinput* event should always be used alongside the *input* event, even though the two events serve distinct roles and are not always used together in typical web implementations. Similarly, for the *macOS Session Resume Support* feature, 16% of test cases were rejected, not due to invalidity but because they addressed general session restoration instead of focusing specifically on Firefox behaviors following an update. A recurring issue was the LLM's tendency to misidentify the target platform. In the *Back Button Intervention feature*, six test cases were rejected as invalid because they were written for Firefox on Android instead of Desktop, incorrectly referencing the SHIP on Android feature. While technically reasonable, these test cases were irrelevant in the given context. Furthermore, 8% of the test cases from all features were rejected for reasons beyond the testing team's scope, with the most frequent issue being the inclusion of performance testing. Examples included scenarios evaluating feature behavior under different network conditions or assessing system performance after session resume—tests that, while useful, were outside the scope of the functional testing phase.

LESSONS LEARNED

LLMs Can Generate Novel Test Plans

The 27% acceptance rate of the test cases in LLM-generated test plans across the eight features shows promising results in generating novel, valid and relevant test cases. This suggests that LLMs can identify testing scenarios that QA engineers might overlook and potentially improve test coverage. While the results show promise in test plan generation, further development is needed to improve the capabilities of LLMs to generate more novel test scenarios effectively. Integration with existing test management systems could help by parsing the manually generated test plans to the LLM and prompting it only to generate the missing test cases, while leveraging feedback loops from QA engineers could further enhance the diversity and novelty of the generated cases.

LLMs Replicate Existing Test Patterns Effectively

The high proportion of "Already Exist" cases (50.5%) indicates that, after learning the general structure and style of Mozilla QA test plans from the few-shot examples, the LLM was able to generate valid and relevant test cases that closely align with those in test plans created by human testers. While these duplicate test cases did not add new test scenarios, they suggest the model effectively internalized the structure, functionality, and scope typical of real QA test plans for the evaluated features. This capability may be especially useful in situations where QA teams have not yet manually created test plans for a new feature. In such cases, the LLM could serve as a starting point by generating an initial test plan with a set of test cases for QA engineers to review and refine. This would potentially save significant time compared to creating test plans from scratch. Additionally, when evaluating the *MacOS Session Resume Support feature*, the QA engineers preferred the LLM's clearer and more structured phrasing over some of their original test plans. This suggests that beyond generating test scenarios, the LLM can serve as a valuable tool for QA engineers to improve test documentation by making it clearer and more structured before distributing it to their testing teams.

Accuracy Depends on Feature Clarity and Context

The relatively low rate of invalid test cases (14.5%) indicates that the LLM generally understands most feature details well. This suggests that LLMs can effectively process and apply information when the features are clearly defined. However, the occurrence of technical misunderstandings such as confusion around the "beforeinput" event in the *Text Input Event* feature highlights the importance of clear feature descriptions and domain-specific knowledge. It also shows the critical role of carefully chosen examples in the prompt. Features like *Text Fragment* and *macOS Session Resume Support*, which have concrete and observable behaviors, saw fewer irrelevant test cases, likely due to their alignment with the provided example test cases in the prompt. In contrast, more abstract or technically complex features, such as *Back Button Intervention*,

led to higher rates of invalid and irrelevant test cases.

LLMs Can Enhance, but Not Replace, QA Efforts

For software and QA teams, the success in generating some novel, valid test cases suggest broader applications in software testing. LLM-generated test plans could be used to enhance existing test plans or serve as a foundation to be refined as needed. Moreover, the preference for LLM-generated phrasing in some cases suggests potential applications in test plan documentation improvement. Based on the observed results and lessons learned, our experience with GPT 4-Turbo tells us that while LLMs can serve as valuable complementary tools to enhance the existing QA process, they should not be viewed as a complete replacement for human expertise in test plan generation.

THREATS TO VALIDITY

In our study, we evaluated our approach on only eight Firefox features representing all features under active development at the time. The various types of feature functionality and complexity provide a broader perspective on our approach's applicability across different scenarios. Furthermore, focusing on actively developed features allowed Mozilla's QA engineers to assess the approach in realistic conditions and provide feedback on its benefits within their active process, which would not be evident if applied to completed features. This feedback is valuable for improving future versions of our approach, and we plan to expand our study to include additional features and systems.

Another significant threat relates to prompt engineering, as the structure and content of prompts significantly influence the LLM's output. We addressed this by iteratively refining our prompt template through multiple iterations, testing different versions to optimize the quality and relevance of the test cases in those generated test plans.

The few-shot examples were drawn from the test plans for three unrelated Firefox features. While this avoided any direct overlap with the features under evaluation, these examples might have biased the model toward producing outputs that reflect their structure, tone, or scenario types.

Additionally, we found that a temperature setting of 0.8 yielded test plans that balanced novelty and relevance. Using different temperature settings might lead to different results. Therefore, further experimentation is needed to assess the impact of varying temperature settings across different domains, scenarios, and levels of feature complexity.

For each feature, our approach was evaluated by a lead QA engineer, which may introduce some degree of subjectivity into the assessment. Nevertheless, the assigned QA engineer was the team lead directly responsible for that feature at Mozilla and was therefore thoroughly familiar with its requirements and existing test plans. To mitigate this threat, the first author also manually evaluated the generated test cases for each feature.

CONCLUSION

In this paper, we shared our experience using an LLM (GPT-4 Turbo) to generate QA test plans with high-level test cases for eight Firefox features. The LLM generated novel test scenarios in 27% of cases and reproduced existing ones in 50.5%.

GPT-4 Turbo performed well on broad, user-facing features where the context was rich and usage patterns were more predictable. Additionally, some of the test cases from the LLM-generated test plans were also phrased more clearly than those written manually by the QA team. In contrast, it struggled with abstract or narrowly scoped features. Across all features, 14.5% of the test cases were invalid, and 8% were out of scope. These issues point to the importance of clear feature descriptions and carefully constructed prompts.

Overall, LLMs like GPT-4 Turbo show strong potential in the QA domain. While not a replacement for human expertise, they can effectively complement human reasoning, improve documentation clarity, and surface missed testing scenarios. In the future, we plan (and encourage others) to evaluate the performance of different LLMs (e.g., Claude and Gemini) on test plan generation. Furthermore, we plan to evaluate our approach on other open-source projects.

ACKNOWLEDGEMENTS

We would like to thank the Firefox QA team for their valuable support throughout this project. In particular, we are grateful to **Tania Maity**, **Tracy Walker**, and **Ben Chatterton** for their assistance in evaluating our approach and their insightful feedback.

References

1. H. Krasner, "The cost of poor software quality in the us: A 2020 report," *Proc. Consortium Inf. Softw. QualityTM (CISQTM)*, vol. 2, 2021.
2. J. Wang, Y. Huang, C. Chen, Z. Liu, S. Wang, and Q. Wang, "Software testing with large language models: Survey, landscape, and vision," *IEEE Trans-*

actions on Software Engineering, vol. 50, no. 4, pp. 911–936, 2024.
3. M. Schäfer, S. Nadi, A. Eghbali, and F. Tip, "An empirical evaluation of using large language models for automated unit test generation," 2023.
4. S. Alagarsamy, C. Tantithamthavorn, and A. Aleti, "A3test: Assertion-augmented automated test case generation," *Information and Software Technology*, vol. 176, p. 107565, 2024.
5. A. Mastropaolo, N. Cooper, D. N. Palacio, S. Scalabrino, D. Poshyvanyk, R. Oliveto, and G. Bavota, "Using transfer learning for code-related tasks," *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 1580–1598, 2023.
6. G. Ye, Z. Tang, S. H. Tan, S. Huang, D. Fang, X. Sun, L. Bian, H. Wang, and Z. Wang, "Automated conformance testing for javascript engines via deep compiler fuzzing," in *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, PLDI 2021, (New York, NY, USA), p. 435–450, Association for Computing Machinery, 2021.
7. S. Hashtroudi, J. Shin, H. Hemmati, and S. Wang, "Automated test case generation using code models and domain adaptation," *CoRR*, vol. abs/2308.08033, 2023.
8. C. Lemieux, J. P. Inala, S. K. Lahiri, and S. Sen, "Codamosa: Escaping coverage plateaus in test generation with pre-trained large language models," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pp. 919–931, 2023.
9. V. Li and N. Doiron, "Prompting code interpreter to write better unit tests on quixbugs functions," 2023.
10. N. Rao, K. Jain, U. Alon, C. L. Goues, and V. J. Hellendoorn, "Cat-lm training language models on aligned code and tests," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 409–420, 2023.
11. U. Mukherjee and M. M. Rahman, "Employing deep learning and structured information retrieval to answer clarification questions on bug reports," 2023.
12. K. K. Shahandashti, A. B. Belle, M. M. Mohajer, O. Odu, T. C. Lethbridge, H. Hemmati, and S. Wang, "Using gpt-4 turbo to automatically identify defeaters in assurance cases," in *2024 IEEE 32nd International Requirements Engineering Conference Workshops (REW)*, pp. 46–56, 2024.
13. D. Li and L. Murr, "Humaneval on latest gpt models – 2024," 2024.
14. B. Li, W. Wu, Z. Tang, L. Shi, J. Yang, J. Li, S. Yao, C. Qian, B. Hui, Q. Zhang, and Z. Yu, "Prompting large language models to tackle the full software development lifecycle: A case study," 2024.
15. J. L. Fleiss and J. Cohen, "The equivalence of weighted kappa and the intraclass correlation coefficient as measures of reliability," *Educational and psychological measurement*, vol. 33, no. 3, pp. 613–619, 1973.

John Pangas is a master's student in the Department of Electrical and Software Engineering at the University of Calgary, Canada. His research interests include AI for Software Engineering (AI4SE), mining software repositories, and software security. He received his bachelor's degree in Software Engineering from Zhejiang University of Technology, China. You can contact him at john.pangas@ucalgary.ca.

Suhaib Mujahid is a Staff Machine Learning Engineer at Mozilla, where he leads projects that address software engineering challenges and bridge the gap between innovative research and practical engineering processes. His research interests include mining software repositories, software ecosystems, and machine learning on code. He received his Ph.D. degree in Software Engineering from Concordia University. Contact him at smujahid@mozilla.com .

Ahmad Abdellatif is an Assistant Professor with the Department of Electrical and Software Engineering, University of Calgary, Canada. His research interests include software chatbots, mining software repositories, and Software Engineering for Artificial Intelligence (SE4AI). He received his Ph.D. degree in Software Engineering from Concordia University. More about his work is available at https://aabdllatif.github.io/.

Marco Castelluccio is a Senior Engineering Manager at Mozilla, where he leads the CI and Quality Tools, Engineering Workflow, PDF, and Translations teams. His research interests include software engineering, software reliability, and artificial intelligence. He received his Ph.D. degree in Computer Science Engineering from the University of Napoli Federico II. You can reach out to him at mcastelluccio@mozilla.com.