LARGE LANGUAGE MODELS, THE NEW SCRUM MASTERS

J. Ortiz Couder, O. Ochoa

Embry-Riddle Aeronautical University (UNITED STATES)

Abstract

Agile is an approach to software development that emphasizes flexibility, collaboration, and customer feedback. It focuses on iterative development cycles, where requirements and solutions evolve through the collaborative effort of self-organizing and cross-functional teams. Scrum is a framework used within agile software development but has found application in various other fields as well. It provides a structure for teams to collaborate effectively on complex projects, allowing them to adapt to changes quickly and deliver high-quality products efficiently. New software engineers need to learn to work in teams with other engineers in complex systems. One method to do so is through capstone classes. These capstone classes tend to be led by one or two professors who must take care of at least 5 groups each. In this paper we propose the usage of Large Language Models (LLMs) as Scrum Masters for the groups to light up the load work of professors. LLMs are a type of artificial intelligence (AI) model designed to understand and generate human-like text. These models are built using deep learning techniques, particularly using architectures like transformers, and are trained on vast amounts of text data to learn patterns, relationships, and language structures. Due to the vast amount in knowledge contained within LLMs and their natural language understanding, the development team and product owners could utilize LLMs as their Scrum Master. Scrum Masters oversee facilitating the Scrum process, removes impediments, and ensures the team adheres to Scrum principles and practices. In this paper we show an approach that, through the use of template queries, allows LLMs to capture the work of a Scrum Master during the daily standup meetings. In our approach, every member of the development team will answer the three daily standup questions to the LLM. The LLM will condense the information, e.g., if a developer is facing an issue that is blocking their progress, the LLM will create a concise report of this. In this work we show how the LLM can point out a potential solution. The professor gets reported on the progress and on anything preventing their progress which the LLM cannot assist on solving. Even though there are still some things that LLMs cannot comprehend nor provide such as empathy, or leadership skills, LLMs may have the necessary technical skills required to successfully carry out Scrum Master related tasks. The main issue LLMs face is that there are some physical impediments that LLMs cannot directly overcome. However, by using the proposed approach, could save time for professors to directly focus on fixing other issues the students are facing.

Keywords: Large Language Model, Scrum, Education, Project Management, Software Engineering, Agile, Scrum Master.

1 INTRODUCTION

In the ever-evolving landscape of project management, the Agile methodology has emerged as a beacon of efficiency and adaptability. At the heart of Agile lies the Scrum framework, providing structure and flexibility for teams to navigate complex projects seamlessly [1]. Scrum is a specific implementation of Agile principles and practices. While Agile provides the overarching philosophy and values, Scrum offers a structured framework with defined roles, ceremonies (such as Sprint Planning, Daily Stand-ups, Sprint Reviews, and Retrospectives), and artifacts (such as the Product Backlog, Sprint Backlog, and Increment) to guide teams in executing Agile principles effectively [2]. There are three primary roles within the Scrum framework; Scrum Master, servant-leaders responsible for ensuring that the Scrum process is followed properly, Product Owner, person who represents the stakeholders and is responsible for maximizing the value of the product by managing the Product Backlog, and, Development Team, which consists of professionals who work together to deliver increments of potentially shippable product functionality during each Sprint. Traditionally, Scrum Masters have been human facilitators, guiding teams through the intricacies of Agile processes. However, as technology continues to advance, the integration of Language Models (LMs), particularly Large Language Models (LLMs), presents a novel and compelling opportunity to revolutionize the role of the Scrum Master. One of the fundamental responsibilities of a Scrum Master is to foster communication and collaboration within the team [3]. LLMs can augment this aspect by analyzing team communications, identifying patterns, and offering insights to enhance team dynamics. For instance, by parsing through Slack messages, emails, and meeting

ISBN: 978-84-09-62938-1

transcripts, LLMs can discern trends in communication breakdowns or areas where clarification is required, enabling Scrum Masters to intervene proactively.

Large Language Models (LLMs) are sophisticated artificial intelligence models designed to process and generate human-like text based on vast amounts of training data [4]. Specifically, they are a type of neural network-based model designed to process and generate human-like text. LLMs utilize deep learning architectures, particularly transformer-based architectures. These models, often built using deep learning techniques such as neural networks, have millions or even billions of parameters [5]. They can understand and generate text in natural language with a high degree of fluency and coherence.

LLMs stand out as computational models renowned for their capacity to accomplish general-purpose language generation along with other tasks in natural language processing, including classification [6]. LLMs have demonstrated remarkable capabilities in understanding and generating human-like text across a diverse range of domains [7]. Their proficiency in natural language understanding, coupled with their ability to process vast amounts of data, equips them with the potential to excel in roles requiring communication, coordination, and problem-solving—qualities inherent to effective Scrum Masters. LLMs have been used for a huge variety of uses related to the software development process; from code generation [8], to code explanation [9], or comparing if pieces of code meet the software requirements specifications agreed between the development team and the customer [10], among other uses

Moreover, the iterative nature of Agile development necessitates constant feedback loops. LLMs can play a pivotal role in providing timely feedback to team members based on their interactions and outputs [11]. Through sentiment analysis and language comprehension, LLMs can gauge the morale of the team, detect potential bottlenecks, and suggest adaptive strategies to maintain project momentum. The Scrum Master serves as a coach, guiding team members in adopting Agile principles and practices. LLMs can serve as repositories of knowledge, offering on-demand guidance and training resources tailored to the team's specific needs [12]. By synthesizing vast repositories of Agile best practices, case studies, and troubleshooting guides, LLMs can provide comprehensive support to teams, empowering them to navigate challenges autonomously.

Critics may express concerns about the ability of LLMs to comprehend the nuances of human interaction and context-specific challenges. However, recent advancements in natural language processing, coupled with fine-tuning techniques, have significantly enhanced LLMs' contextual understanding and adaptability [13]. The integration of LLMs as Scrum Masters represents a paradigm shift in Agile project management. By harnessing the capabilities of these advanced language models, organizations can streamline communication, enhance productivity, and foster a culture of continuous improvement within Agile teams. While challenges and ethical considerations remain, the potential benefits of leveraging LLMs in Scrum Master roles are too compelling to ignore, heralding a new era of AI-enabled Agile project management.

In this paper we propose that LLMs, more specifically GPT, can be used as Scrum Masters in a classroom setting, to aid professors by relieving the load of having to be Scrum Masters for multiple capstone groups. By doing this, the professors can focus on dealing with the most important issues, that LLMs can't aid by fixing them. The main reason why GPT-3.5 was the used LLM is because it is one of the most widely used LLM [14].

The organization of this paper is as follows. Section I introduces the main issues with existing capstone classes, and why LLMs could prove a potential solution. Section II displays the followed methodology to test if LLMs can indeed be used as a tool to decrease instructor's workload. Section III shows the results and Section IV contains the authors' suggested future work and concluding statements.

2 METHODOLOGY

As stated earlier, Scrum is a framework used in Agile project management that emphasizes iterative and incremental development to deliver high-value products efficiently [1]. One of the most important ceremonies that occur during the development of a product are daily standup meetings. They are usually 15-minute-long meetings where each member of the development team meets with the scrum master and answers three questions: What did you do since the last scrum meeting? Do you have any obstacles? What will you do before the next meeting? [15].

These three questions set the foundation that the paper will utilize. A template is used that allows the developers to write the answers for each question and notify GPT on their answers. If there are any obstacles to which GPT can provide guidelines on how to solve, it will do so. However, if there are obstacles to which it doesn't have a solution to or is unable to find the solution by itself, that obstacle is GPT then gets all the information and tells it to the Scrum Master later. Once the team is done with their standup meeting, the professor will interact with GPT. This is when the professor asks for any updates regarding the team's progress. If everything is going smoothly or there were some obstacles GPT could help fix, GPT will simply update the professor on that progress, yet if there are obstacles GPT could not help with, the professor will be notified about this.

Student 1 (Frontend Developer):

1. Yesterday, I implemented the user registration page.
2. Today, I plan to implement the login page.
3. No obstacles.

Student 2 (Backend Developer):

1. Yesterday, I set up the database schema for user accounts creation.
2. Today, I'll set up the database schema for user account authentication.
3. No obstacles.

Student 3 (UI/UX Designer):

1. Yesterday, I finalized the wireframes for the parking spot selection screen.
2. Today, I'll start working on high-fidelity mockups for the parking spot reservation flow.
3. No obstacles.

Summary for Main Scrum Master:

- Student 1 is working on implementing the login page after completing user registration.
- Student 2 progressed with backend tasks, focusing on database schema for user account authentication.
- Student 3 continued with UI/UX design, moving to high-fidelity mockups.
- No obstacles reported by any team member.

Figure 1. Example of standup meeting's notes taken by GPT.

Fig. 1. displays the summary of the interaction between a group of three developers and GPT. In this case, the project is an app that aims to provide remote selection of parking spots for other students on campus and reduce the time it usually takes to park in school. The development is still in the early stages and there are no obstacles to be reported yet. However, Fig. 2. below displays a scenario when there is indeed an obstacle. In this situation, Student 1, who oversees the Front-end of the app, is having issues with the reservation of the parking spots. It seems like there might be a contradiction between two customer requirement specifications, one which requires the selected spot to be displayed within 250 milliseconds, and one that made the refresh rate of the page to be 500 milliseconds to maintain a low energy consumption.

Student 1 (Frontend Developer):

1. Yesterday, I tried to show in the parking map the selected spots by the user within the 250-millisecond constraint given by the customer, but since we are refreshing the page every 500 milliseconds, it is not possible to do so.
2. Today, I will try to fix this issue.
3. The ping to the server is every 500 milliseconds to meet the energy consumption requirement made by the customer, but the customer also wanted to display the selected spot within 250 milliseconds. They are mutually exclusive so one has to be modified.

Student 2 (Backend Developer):

1. Yesterday, I set up the parking spot blocking call. Now if a user has selected a parking spot, that spot's status is set to unavailable so no other user can select it.
2. Today, I will work on setting the server confirmation response when that spot has been successfully selected.
3. No obstacles.

Student 3 (UI/UX Designer):

1. Yesterday, I designed the parking spot selection screen. Now if a user has selected a parking spot, that spot will be displayed as yellow until it is confirmed reserved, when it will change to red.
2. Today, I will work on the reservation confirmation message displayed.
3. No obstacles.

Summary for Professor (Main Scrum Master):

- Student 1 (Frontend Developer) encountered challenges with displaying selected spots within the time constraint and the conflicting server ping frequency requirement. They will work on resolving these issues.
- Student 2 (Backend Developer) set up the parking spot blocking call and will work on server confirmation responses today.
- Student 3 (UI/UX Designer) designed the parking spot selection screen and will work on the reservation confirmation message today.
- No obstacles reported by any other team member.

Figure 2. Example of standup meeting's notes where a student reports an obstacle taken by GPT.

When GPT could envision a solution to an obstacle, it would instantly describe it to the student, instead of doing so after the standup meeting. This is something that could become troublesome as it could extend the duration of standup meetings. In the scenario when GPT could not instantly find a potential solution to an obstacle a student faced, it would not mention anything and simply report it to the professor at the post-standup meeting, as shown in Fig. 3. This figure displays the post-standup meeting between the professor and GPT. In this case, since there is an obstacle to which GPT could not provide a solution, it asks the professor for assistance in resolving the conflicts between the two contradictory requirement specifications by potentially meeting with the customer and discussing potential changes in those requirements.

Post-Standup Meeting:

Professor:
How did the standup meeting go today?

GPT Scrum Master (Scrum Master for Developers):
Overall, it went well. However, Student 1 encountered challenges with the time constraint and conflicting requirements, while Student 3 made good progress with their tasks.

Professor:
I see. Do you require any assistance in resolving those challenges?

GPT Scrum Master:
Yes, indeed. We'll need your guidance to navigate through the conflicting requirements and find a suitable solution.

Professor:
Of course, I'll be happy to help. Let's discuss the details and come up with a plan to address these challenges.

GPT Scrum Master:
Great, thank you!

And that concludes our morning standup! If you have any further questions or need additional information, feel free to ask.

Figure 3. Post-standup meeting where GPT reports to the professor an obstacle to which it cannot find a solution.

3 RESULTS

In total, six groups of students were used to evaluate this procedure. Each of the groups had a different project to conduct but all had the same professor overseeing those projects.

Table 1. Summary of obstacles reported and solutions by GPT.

	Obstacles reported	***Obstacles to which GPT provided a potential solution***	***Obstacles to which GPT's provided solution worked***	***Obstacles to which GPT could not provide a solution***
Group 1	25	15	11	10
Group 2	33	20	15	13
Group 3	21	16	10	5
Group 4	15	8	5	7
Group 5	35	30	20	55
Group 6	29	25	22	4
Total	158	114	83	44

In total, 158 obstacles were reported, and GPT gave a solution to 114 of them. However, out of those 114, only 83 proved to be successful in fixing the obstacles, meaning that GPT gave a solution to 72% of the problems and gave a successful solution to 52.5% of the total reported obstacles. In total, GPT could not provide a solution to 44 (27.8%) obstacles and required the assistance of the professor to help the teams with the hurdle. Between the solutions GPT provided that didn't solve the obstacle, and the ones GPT couldn't find the solution to, total of 75, GPT was incapable of solving 47.5% of the obstacles reported.

It is true that each of the projects was different with a different level of difficulty, which could affect the number of obstacles each group faced, and how easy it was for GPT to provide solutions. The skill level of each group is also varying, which again could vary the number of obstacles found. If two groups, one of highly skilled developers, and one of less skilled developers tackled the same project, it is to be assumed the one with the most skilled developers would encounter less obstacles they couldn't solve themselves. This discrepancy in skill levels also means that GPT was tasked with solving more complex issues and would struggle more to do so.

Table 2. Categorization of obstacles found.

Category	***Obstacles***	***Solved***
Logical errors	60	53
Contradicting/incorrect requirement	12	0
Compatibility issue	25	10
Need of new/different hardware	33	0
Integration between front-end and back-end	28	20
Total	158	83

Table 2 displays the different categories of the obstacles reported. There are five total categories: logical errors, contradicting or incorrect requirements, compatibility issues with the hardware, need of a new or different hardware, and integration between the front-end and the back-end. Logical errors are problems that occur when the code doesn't behave as expected [16]. They are usually due to logic errors. The code compiles and runs yet the behavior is not the expected one. There were 60 reported logical errors to the code and GPT could solve 53 by giving the entering the code and asking it to debug the code. Compatibility issues seemed to be another common issue encountered. When developing a mobile application for instance, all different hardware must be considered and might require different interfaces. There were 25 reported compatibility issues reported and GPT could solve 10. The integration between the front-end and back-end seemed to be another common obstacle. These obstacles tend to happen when the communication between the part of the application the user interacts with and the one that makes the program work correctly is broken [17]. GPT could fix 20 out of the 28 reported integration issues. When looking at these three categories, GPT could fix 83 of 113 (73.5%).

However, there were two categories in which GPT could not provide any assistance with. These two were contradictory or incorrect requirements and when the new hardware was required. A total of 40 obstacles were reported and GPT was incapable of fixing any of them. This also aligns with the idea that GPT needs a person to oversee its work. In these cases, either communication with the customer was required, to fix or modify the requirements, or some sort of interaction with vendors was required to order hardware. Since GPT was not allowed to perform said actions, it relied on the professor overseeing the projects to handle them instead.

4 CONCLUSIONS

This paper suggests that LLMs could be used to lighten the load professors have when teaching capstone classes focused on group projects. This paper proves that GPT can be utilized to solve obstacles development teams encounter, yet there are some problems that require direct interaction with external sources – customers or vendors – which GPT cannot assist in solving. If issues like these are encountered, then a professor (or supervisor) is required to help overcoming them. Yet, if the issues encountered are strictly code related, GPT could be a potential aid to solve the problems so the professor can focus on other obstacles.

It is worth noting that this paper does not believe LLMs should be a replacement for other developers during these activities but a tool that could aid during them. While our research offers promising results, future studies might consider evaluating different LLMs and versions within those LLMs against the same projects and obstacles to compare how capable each version is at solving those reported obstacles.

REFERENCES

[1] A. Srivastava, S. Bhardwaj and S. Saraswat, "SCRUM model for agile methodology," in *2017 International Conference on Computing, Communication and Automation (ICCCA)*, 2017.

[2] M. Paasivaara, S. Durasiewicz and C. Lassenius, "Distributed agile development: Using scrum in a large project," in *2008 IEEE International Conference on Global Software Engineering*, 2008.

[3] K. Schwaber, "The Scrum Master," in *Agile project management with Scrum*, Microsoft Press, 2004, pp. 36-47.

[4] J. Hoffman, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. de Las Casas, L. A. Hendricks, J. Welbl, A. Clark, T. Henningan, E. Noland, K. Millican, G. van den Driessche, B. Damoc, A. Guy, S. Osindero, K. Simonyan, E. Elsen, J. W. Rae, O. Vinyals and L. Sifre, "Training compute-optimal large language models," *arXiv preprint arXiv:2203.15556,* 2022.

[5] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," *arXiv preprint arXiv:1909.08053,* 2019.

[6] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, E. H. Chi, T. Hashimoto, O. Vinyals, P. Liang, J. Dean and W. Fedus, "Emergent abilities of large language models," *arXiv preprint arXiv:2206.07682,* 2022.

[7] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, W. Yu, Y. Zhang, Y. Chang, P. S. Yu, Q. Yang and X. Xie, "ASurvey on Evaluation of Large Language Models," *ACM Transactions on Intelligent Systems and Technology,* vol. 15, no. 3, pp. 1-45, 2024.

[8] J. Liu, C. S. Xia, Y. Wang and L. Zhang, "Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation," *Advances in Neural Information Processing Systems,* vol. 36, 2024.

[9] S. Sarsa, P. Denny, A. Hellas and J. Leinonen, "Automatic generation of programming exercises and code explanations using large language models," in *Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 1*, 2022.

[10] J. Ortiz Couder, D. Gomez and O. Ochoa, "Requirements Verification Through the Analysis of Source Code by Large Language Models," in *SoutheastCon 2024*, 2024.

[11] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakatantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever and D. Amodei, "Language models are few-shot learners," *Advances in neural information processing systems,* vol. 33, pp. 1877-1901, 2020.

[12] F. Petroni, T. Rocktäschel, P. Lewis, A. Bakhtin, Y. Wu, A. H. Miller and S. Riedel, "Language models as knowledge bases?," *arXiv:1909.01066,* 2019.

[13] B. Wang, G. Li and Y. Li, "Enabling conversational interaction with mobile ui using large language models," in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, 2023.

[14] T. Wu, S. He, J. Liu, S. Sun, K. Liu, Q.-L. Han and Y. Tang, "A brief overview of ChatGPT: The history, status quo and potential future development," *IEEE/CAA Journal of Automatica Sinica,* vol. 10, no. 5, pp. 1122-1136, 2023.

[15] K. Schwaber and M. Beedle, Agile software development with Scrum, Prentice Hall PTR, 2001.

[16] A. Ettles, A. Luxton-Reilly and P. Denny, "Common logic errors made by novice programmers," in *Proceedings of the 20th Australasian Computing Education Conference*, 2018.

[17] F. Günzel and A. B. Holm, "One size does not fit all—understanding the front-end and back-end of business model innovation," *International journal of innovation management,* vol. 17, no. 1, p. 1340002, 2013.