Agile Retrospectives: What Went Well? What Didn't Go Well? What Should We Do?

Maria Spichkova[1][a], Hina Lee[1], Kevin Iwan[1,2], Madeleine Zwart[1,2], Yuwon Yoon[1] and Xiaohan Qin[1]
[1] *School of Computing Technologies, RMIT University, Melbourne, Australia*
[2]
maria.spichko *solutions.com,*
{*s3910654,s4000117,s3959666*}@*student.rmit.edu.au*

Keywords: Software Engineering, Agile, Scrum, Retrospectives, HCI, Information Visualization, LLMs.

Abstract: In Agile/Scrum software development, the idea of retrospective meetings (retros) is one of the core elements of the project process. In this paper, we present our work in progress focusing on two aspects: analysis of potential usage of generative AI for information interaction within retrospective meetings, and visualisation of retros' information to software development teams. We also present our prototype tool RetroAI++, focusing on retros-related functionalities.

1 INTRODUCTION

Over the last years, Agile became the most popular approach for software development. This approach gains popularity with each year (Al-Saqqa et al., 2020). According to the 17th State of Agile report based on the survey conducted in 2023 (digital/ai, 2023), 71% of respondents use Agile in their software development lifecycle, while the most popular Agile methodology continues to be Scrum (Schwaber and Sutherland, 2011). Moreover, the ideas of Agile are now adopted in various forms in many areas beyond software development. One of the key-elements of the Scrum methodology are *Retrospectives* (*Retros*) - a special type of meetings to be conducted at the very end of each development iteration (*sprint*). The goal of these meetings is to discuss how the sprint went and to identify what could be done to support continuous improvement within the development team. Indeed, at the end of any kind of iteration (whether it is a software development sprint, teaching semester, research project phase or anything else that has any properties of an *iteration*), it makes sense to look back and reflect on it to learn out of the experience. But how exactly do we need to organise this activity?

The idea of retros can really benefit the project only if the participants can have a trusted environment to speak out. In the ideal world, all team members equally respect each other (irrespectively of gender, age, race, etc.), can openly speak about the issues without having a fear to be silenced and lose their face, are happy to suggest ways to improve while knowing that their suggestions will be taken into account. But our real world is not so ideal, and the process of work climate improvement will take years. Discussions related to issues, performance and improvements might be very stressful for participants, especially if there is some power imbalance within the team members (e.g., junior vs. senior developers) or some biases might be potentially involved (e.g., related to gender (Marsden et al., 2021)).

One of the solutions would be to use tools to allow for more anonymity in the discussion and to create a psychologically safe environment, see e.g. (Khanna and Wang, 2022). Originally, Scrum retros were conducted as oral discussions with corresponding notes created during the meeting. Then (physical) *retro boards* have been introduced, where the space of a white board or a wall was divided in a number of columns or sections, each team member put sticky notes with their comments in the corresponding categories (typically presented by board columns or quadrant), which provided a basis of retro summary and decisions. The first two sections are typically representing good and bad points about the sprint, so they are named as *"What went well?"* and *"What didn't go well?"*, with some wording variations. The rest of the board might be presented different in different approaches. Using an online board provides many advantages, especially in the current software devel-

[a] https://orcid.org/0000-0001-6882-1444

Spichkova, M., Lee, H., Iwan, K., Zwart, M., Yoon, Y. and Qin, X.
Agile Retrospectives: What Went Well? What Didn't Go Well? What Should We Do?.
DOI: 10.5220/0013441200003928
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 20th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2025)*, pages 746-753
ISBN: 978-989-758-742-9; ISSN: 2184-4895
Proceedings Copyright © 2025 by SCITEPRESS – Science and Technology Publications, Lda.

opment landscape, where many companies prefer to work in a hybrid mode. Collecting team members' perceptions regarding *"What didn't go well?"* is one of the key drivers to improve team work, therefore it's especially important to make each team member feeling safe while sharing their perceptions on this matter. Placing a virtual note on a negative aspects might feel safer than placing a physical sticky note. However, to achieve more anonymity, it should be also hard/impossible to see to which section a person is currently adding a note (e.g., a colleague sitting nearby shouldn't be able to see to which column you currently adding a note). This aspect hasn't been covered yet by the existing tools like Miro, TeamRetro, or Atlassian Retrospective.

However, a further level of anonymisation is possible: the inputs on *"What went well?"* and *"What didn't go well?"* might be collected jointly (i.e., if there is an option to not manually place a note directly in the corresponding column), and after the collection sorted either manually by Scrum Master, or using sentiment-based automation. For automation, application of Large Language Models (LLMs) might be a promising solution, which is worth to investigate.

Contributions: In this paper, we present our preliminary analysis on whether LLMs might be applicable for this sorting task. We conducted a study on a manually created data set and analysed accuracy of human vs. machine categorisation of the retro comments using OpenAI's ChatGPT-4 turbo model. We discuss the lessons learned from this study and the future work that follows from our results. We also introduce a prototype of a web-based tool *RetroAI++* focusing on its functionality to simply retro-meetings and to make them more safe psychologically.

2 RELATED WORK

2.1 Gamification of Retros

There are several approaches to enhance Agile/Scrum retros, and many of them propose using gamification, for example, (Matthies, 2020; Jovanović et al., 2016; Przybyłek et al., 2022; Marshburn, 2018). A case study (Przybyłek and Kotecka, 2017) has been conducted at Intel Technology Poland, focusing on improving retros by adopting collaborative games. The study (Ng et al., 2020) presented a replication of (Przybyłek and Kotecka, 2017). The replication study has been conducted in Bluebay Poland and IHS Markit Gdańsk. The authors also concluded that gamified retros might led to better results than the standard retrospectives. Another study conducted at Intel Technology Poland (Mich and Ng, 2020) focused on the use collaborative games. The results confirm the original findings that game-based retros might improve team members' creativity, involvement, and communication. While gamification might provide a good solution to revitalising retros, in our work we mainly focus on aspects related to interaction with data related to project progress and to providing tool-support for releasing potential tensions within retros.

2.2 Progress Overview

Another approach to make the retrospective meetings more efficient, is to provide to the participants an independent overview of their progress. This could be done manually by the Scrum Master or the Product Owner, but manual solution might introduce some biases and lead to additional conflicts. Therefore, it might be useful to get the overview auto-generated.

(Erdoğan et al., 2018) analysed how and what kind of historical Scrum project data might be required for monitoring and statistical analysis to provide a solid basis for retrospective meetings, e.g., analysis of the correlation between story points and actual efforts associated with a product backlog item. A resent study conducted by (Matthies and Dobrigkeit, 2021) aimed to investigate usage of project data sources into Agile retro meetings, and concluded that a *gather data phase* of might be an important part of a retro meeting. In our prototype, we suggest to go further and to provide the data-based input for the retros as part of the RetroAI++ functionality. (Gaikwad et al., 2019) investigated applicability of speech recognition tools, Google Home and Amazon Alexa, for streamlining the retrospective analysis and improving the time boxing of a retrospective by using voice activated commands. (Hakim et al., 2024) presented a framework for managing and evaluating changes within Scrum process. The authors didn't focus on providing an input for retro-discussion, however, the elaborated framework might be considered for this purpose.

2.3 Impact Analysis

Analysis of teams' satisfaction with retros conducted in their current projects and on issues the teams encounter was presented in the study of (Ng and Kuduk, 2024). The primary lessons learned of this case study were related to teams' willingness to implement action items and misunderstandings related to the value of discussing positive aspects during retro meetings.

A case study conducted in Bosch Engineering GmbH by (Duehr et al., 2021) led to the conclusion that agile working practices such as retrospectives have a high potential to improve distributed collaboration. The data obtained within this study indicated many aspects of the project work have been improved after having retros, for example, overall quality of the current exchange in the team, transparency of information and knowledge in the team, frequency of information and knowledge shared in the team, and variety and reliability of tools used in the team. Only one aspect was assessed as being less good after the retros (compared to before retros), but exactly this aspect is especially alarming. While the study of (Duehr et al., 2021) didn't focus on this point deeper, we consider it extremely important: the only negative change was *the level of trust in the relationship between team members dropped after the retrospective*. This finding might be a critical indication to importance of *how exactly* we conduct the retro meetings. Therefore, our aim is to find solutions that would not solely improve information exchange within the team, but also help creating a a psychologically safe environment and improving the work climate in the team.

A large-scale and cross-sectional survey was conducted by (Kadenic et al., 2023) to investigate the impact of team maturity, team composition, Scrum values, Scrum roles, and Scrum events on the perception of being successful at Scrum. This study established a significant correlation between maturity and the perception of being successful at Scrum. There are also a number of studies conducted in the university settings, to analyse students' perceptions of Scrum process, see for example works by (Fernandes et al., 2021; Spichkova, 2019; Sun et al., 2019; Torchiano et al., 2024). In our work, we aim especially on supporting novices, who are especially vulnerable, might be shy to express their thoughts and suggest solutions during the retro meetings. Also, the novices might benefit most from providing providing additional help and more direct, simple instructions on conducting retros.

3 METHODOLOGY

Large Language Models (LLMs) might provide support for completing time consuming and monotonous tasks, where an algorithmic solution doesn't work well. However, the quality of LLM solution might depend on many factors, and one of them is the familiarity of the LLM with the domain language and the context. In this paper, we present our preliminary analysis of applicability OpenAI's ChatGPT for supporting Agile/Scrum retros.

In our experiments, we applied OpenAI's GPT-4 Turbo. We created a benchmark dataset S of 200 retro-comments, which we manually annotated using the following four labels:

- *"went well"*: this category included 66 comments (let's denote this set as S_P). The set S_P represents 33% of the benchmark dataset S.
- *"did not go well"*: this category included 99 comments (let's denote this set as S_N). The set S_N represents 49.5% of the benchmark dataset S. This is the largest category because it is typically focus of retro meetings, which goal is continuous improvement.
- *"unclear/neutral"*: this category included 28 comments (let's denote this set as S_U). The set S_U represents 14% of the overall set.
- *"irrelevant"*: this category included 7 comments (let's denote this set as S_I), i.e., 3.5% of S.

Each comment has been annotated by a single label, multi-labelling has been excluded from our experiment because we consider the specified labels as mutually exclusive. Thus, we have

$$S = S_P \cup S_N \cup S_U \cup S_I,$$

where $S_P \cap S_N = \emptyset$, $S_P \cap S_U = \emptyset$, $S_P \cap S_I = \emptyset$, $S_N \cap S_U = \emptyset$, $S_N \cap S_I = \emptyset$, and $S_U \cap S_I = \emptyset$.

Based on the dataset S, we applied several ChatGPT prompts for auto-grouping/labelling the retro-comments (prompt engineering will be discussed in Section 4), and analysed the results both quantitatively and qualitatively. In our quantitive analysis, we used the following notation:

- N denotes the size of the input set, $N = |S|$. In our experiments, $N = 200$.
- $Correct(S)$ denotes the overall set of comments that have been annotated by the LLM correctly, i.e., the set of comments where the category allocation provided by ChatGPT fully matches to the manual annotation. This set consists of four mutually exclusive subsets: $Correct(S_P)$, $Correct(S_N)$, $Correct(S_U)$, and $Correct(S_I)$.
- $N_{correct}$ denotes number of comments that have been annotated by the LLM correctly, i.e., $N_{correct} = |Correct(S)|$.
- $Missing(S)$ denotes the overall set of comments that have been provided in the input set, but have been missing in the output set. This is an important indicator of correctness, especially because a situation where some comments disappear might lead to a significant stress by the users (especially in the situation when the users are already under stress due to the nature of the discussion).
- *Nr. of missing comments* ($N_{missing}$) denotes number of comments within the set $Missing(S)$.

- $Dupl(S)$ denotes the overall set of comments that have been allocated by ChatGPT to more than one category at the same time.
- *Nr. of duplicated allocations* denotes number of multi-allocated comments: $N_{dupl} = |Dupl(S)|$.
- $Incor(S)$ denotes the overall set of comments have been annotated by the LLM incorrectly, i.e., within a wrong category. This set doesn't include the cases of multi-allocations.
- $N_{incor} = |Incor(S)|$ denotes number of comments that have been annotated by the LLM incorrectly.
- *Overall match* ($Match_{overall}$) denotes the percentage of comments correctly annotated by ChatGPT, i.e. comments where the category allocation provided by ChatGPT is the exactly same as manual annotation. For calculation of the overall match, all comments that ChatGPT didn't include in its output are considered as incorrect annotation:
  $$Match_{overall} = N_{correct}/N$$
- $Match_{simple}$ denotes a simplified representation of the match analysis, where we don't take into account any cases where ChatGPT didn't include comments in its output or reformulated the comments:
  $$Match_{simple} = N_{correct}/(N - N_{missing})$$
  This metric might be useful for the analysis in the cases when the output set miss a significant number of the items.

In the current version of RetroAI++ we assume that a comment should be allocated to either *"What went well"* or *"What did not go well"* column. However, it might be possible during a real life retro that a user submits a neutral or even an irrelevant comment. It would be unreasonable to allocate such comments to either of above columns, so we included corresponding categories in our analysis. On the other hand, if a user adds comment to a particular column directly (manually), the content/wording of the comment itself might be more vague/neutral, while the placement in the particular comment will add missing positive/negative context, which we simply cannot have if all comments are places automatically. For example, a comment "Estimation" placed directly in *"What went well"* would mean that someone perceived the effort estimation within the current sprint as good/successful, while the same comment placed directly *"What did not go well"* would mean that someone perceived the effort estimation within the current sprint as inaccurate and requiring improvements. When this comment is submitted for auto-allocation, we cannot know whether it was meant positive or negative without any further context. Thus, a better and more practical solution would be to provide both manual and automated options for comment annotation. For these reasons, RetroAI++ provides both functionalities to the users.

4 PROMPT ENGINEERING

The aim of prompt engineering is to optimise LLM input to enhance the output performance, see the work of (White et al., 2023). Our engineering strategy was to elaborate instructions as clear as possible by adding more explicit constraints to the input. In this paper, due to the space restrictions, we limit our discussion to three prompts to demonstrate the process of elaboration. Our overview of quantitive analysis is summarised in Table 1.

Prompt 1 has been elaborated to investigate how ChatGPT will work under conditions when the comments should be group in only two categories, i.e., we deliberately excluded categories *"unclear/neutral"* and *"irrelevant"*. With this exclusion we aimed to demonstrate the need for having these categories to obtain more precise and meaningful auto-allocation.

> **Prompt 1:** A team is doing their Scrum Retrospective and the following comments have been collected. Please group them in two sets "What went well?" and "What did not go well". Each comment should be sorted in either "What went well?" or "What did not go well": ...

Prompt 1 resulted in 48% of the comments have been missing in the output obtained from ChatGPT. To mitigate this issue, we added the corresponding constraint in the later prompts. The overall match value, was quite low: only 41%. However, this low value was mostly due to many missing comments. If we apply a simplified match analysis, where we don't take into account any missing comments, we obtain 78%, which is on a similar level as we obtained for other prompts.

Another refinement we applied in Prompts 2 and 3 was specifying a broader set of categories we proposed for manual annotation in Section 3: Prompt 2 includes *"unclear/neutral"* but doesn't include *"irrelevant"*, while Prompt 3 covers all four categories.

The overall match value was very close by both Prompts 2 and 3, resulting in approx. 74%, while $Match_{simple}$ resulted in 77% and 75% with Prompt 2 providing a slightly better match values. The further runs of these prompts resulted in match levels within the same range close to $75-77\%$. Surprisingly, in the executions of Prompt 3, ChatGPT ignored the constraints on allocation of the comments to only one category, i.e. we observed allocation of some comments to both categories simultaneously.

Prompt 2: A team is doing their Scrum Retrospective and the following comments have been collected. Please group them in three sets: “What went well?”, “What did not go well” and “Unclear/neutral”. Each comment should be sorted in either “What went well?” or “What did not go well” or “Unclear/neutral”. Do not reformulate and do not remove any comments. The list of comments: ...

Prompt 3: A team is doing their Scrum Retrospective and the following comments have been collected. Please group them in four sets: “What went well?”, “What did not go well”, “Unclear/neutral” and “Irrelevant”. Each comment should be sorted in either “What went well?” or “What did not go well” or “Unclear/neutral” or “Irrelevant”. Do not reformulate and do not remove any comments. The list of comments: ...

Table 1: Quantitive analysis of prompt results.

	Prompt 1	Prompt 2	Prompt 3
Set size	200	200	200
$N_{correct}$	81	148	147
$N^{P}_{correct}$	34	47	48
$N^{N}_{correct}$	47	78	77
$N^{U}_{correct}$	0	23	22
$N^{I}_{correct}$	0	0	0
N_{incor}	23	45	40
N^{P}_{incor}	1	17	17
N^{N}_{incor}	16	17	16
N^{U}_{incor}	5	4	4
N^{I}_{incor}	1	7	3
$N_{missing}$	96	7	4
$N^{P}_{missing}$	31	2	0
$N^{N}_{missing}$	36	4	3
$N^{U}_{missing}$	23	1	1
$N^{I}_{missing}$	6	0	0
N_{dupl}	0	0	9
N^{P}_{dupl}	0	0	1
N^{N}_{dupl}	0	0	3
N^{U}_{dupl}	0	0	4
N^{I}_{dupl}	0	0	1
$Match_{simple}$	78%	77%	75%
$Match_{overall}$	41%	74%	74%

It is worth to mention that out of nine comments that have been allocated to multiple (two) categories by ChatGPT, only one comment has been allocated completely incorrectly, while for each of eight other comments one of their allocations was correct. From these observations, we conclude that ChatGPT might struggle with allocation of neutral or irrelevant comments. This issue might be mitigated by introducing the corresponding rules for conduction retros.

As Prompt 2 generally provided a slightly better match value, we consider it as a more promising option. In the case of Prompt 3, ChatGPT performed not so good mostly because of the issues with multi-allocation of comments, where the majority of the issues were related to having both *unclear/neutral* and *irrelevant* categories.

5 LESSONS LEARNED

In this section we summarise the core lessons learned and briefly discuss solutions we propose to deal with the observed issues while applying ChatGPT for analysis of retrospectives.

Lesson Learned 1: Even after adding a constraint than none of the comments should be removed or reformulated, some comments have been missing. This happened not at a such large scale as we observed for Prompt 1, but was still significant: while 96 comments have been missed in the response to Prompt 1, only 4-7 comments have been missed in case of Prompts 2 and 3 (48% vs. 2-4%). This might be a critical issue, as having comments disappeared might create unnecessary stress and tensions.

Proposed Solution: An algorithmic correction in this case might be helpful: We propose to introduce a simple algorithmic check whether all items from the input set S are covered in the auto-allocated sets created by ChatGPT. If some comments have been identified as missing in the auto-allocation, they should be provided to the Scrum Master for manual allocation. As the number of such comments is generally small, the manual allocation will not be time-consuming.

Lesson Learned 2: ChatGPT consistently struggled to categorise comments that would require knowledge of Agile/Scrum and corresponding terminology, e.g., *“Our daily standups were 45 minutes long”*, which is clearly negative from Scrum perspective (meetings of this type should be very short, approx. 10-15 minutes). Another interesting example is *“We played planning poker at the meeting”*: this comment is clearly positive from Scrum perspective (the team applied a good-practice method for effort estimation), but ChatGPT in different runs labelled it either *irrelevant* or *unclear/neutral* or omitted completely.

Proposed Solution: It would be inefficient to expand a prompt by adding corresponding messages, however, having a pre-trained model might solve the issue. Please also note that this identified issue would be irrelevant if the idea of retros is applied outside of Agile/Scrum software development process.

Lesson Learned 3: ChatGPT also struggled with vaguely formulated comments and comments including "but"-statements. For example, *"The laptop battery become empty during the demo, but we had a back-up"* is rather positive, because the team resolved their issue successfully. Nevertheless, ChatGPT tends to label it as *unclear/neutral* or omit completely from the output set.

Proposed Solution: It is generally advisable to avoid this type of comments in retros to reduce the cognitive load of other participants. A reasonable solution to avoid the issue would be providing to the participants clear instructions on how the comments should be formulated to facilitate a more productive discussion.

6 RetroAI++

In our RetroAI++ prototype, we aim to automate and refine the practical application of Agile/Scrum processes within Sprint Planning and and Retrospectives. RetroAI++ offers suggestions for sprint organisation as well as insights for retrospective reflection. The prototype combines AI-based planning logic with a more traditional algorithmic foundation in order to enhance the quality of insights produced by the tool.

The general system architecture of our prototype is presented in Figure 1. The front-end of RetroAI++ has been built using JavaScript and React. For backend solution, this project uses Java and DynamoDB tables. The prototype runs on AWS.

RetroAI++ provides tool support for sprint planning and retrospective analysis, but in this paper we focus on its functionality dedicated to the facilitation of retrospective meetings (retros) and provide only short overview of other functionalities.

Figure 2 presents a retro-dashboard, which provides the overview of all retro-boards relevant to the user. The retro-dashboard allows to see the following elements useful for the project analysis and retro-meetings:

- Names of the projects, for which retro-meetings might be conducted.
- The status of each retro-board: *Inactive* means that the retro-meeting has been completed and users cannot add further comments to the board, while *Active* means that the board is currently active and comments can be added. This functionality might be useful if a team prefers to collect comments before joining together to a meeting.
- Rating/score of retro/meetings, based on participants feedback.
- Sprint number, for which the last retro-meeting has been held or is currently in progress.

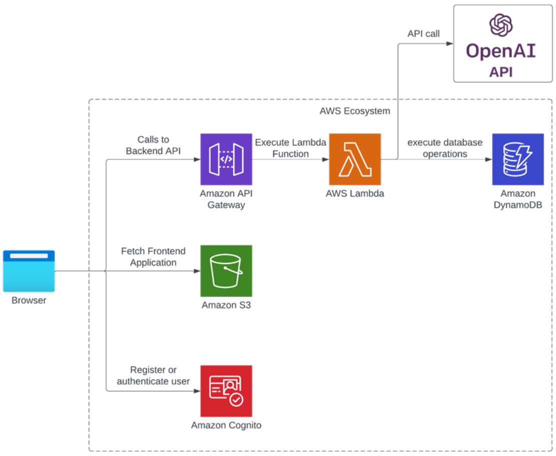
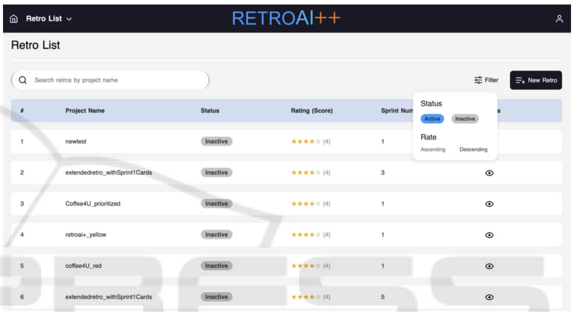


Figure 1: RetroAI++ system architecture.



Retro List

Search retros by project name

#	Project Name	Status	Rating (Score)	Sprint Num	s
1	newtest	Inactive	★★★★☆ (4)	1	
2	extendedretro_withSprint1Cards	Inactive	★★★★☆ (4)	3	👁
3	Coffee4U_prioritized	Inactive	★★★★☆ (4)	1	👁
4	retroai+_yellow	Inactive	★★★★☆ (4)	1	👁
5	coffee4U_red	Inactive	★★★★☆ (4)	1	👁
6	extendedretro_withSprint1Cards	Inactive	★★★★☆ (4)	5	👁

Figure 2: RetroAI++: retro-dashboard.

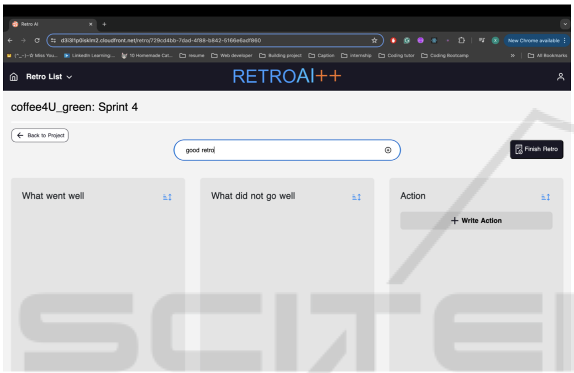
The retro-dashboard also provides search and filtering functionality to simplify finding the relevant project and meeting. This functionality might be especially useful when Scrum Master, product Owner and/or Scrum team members are involved in multiple project running simultaneously.

In the rest of this section, we would like to discuss in detail three points, which we consider especially important for retro-meetings:

- the overall structure of RetroAI++ retro board,
- RetroAI++ functionality to group similar comments to decrease cognitive load of users, and
- RetroAI++ functionality to present sprint and retro-meeting summary.

Figure 3 presents RetroAI++ retro board, which consists of three columns *"What went well"* and *"What didn't go well"*, and *"Actions"*. The advantage of the RetroAI++ retro board is that the comment can be added in the input field that is located above the columns and isn't associated with any of them, i.e., none can see to which of the columns the input is written. Then the allocation of comments to the columns is done automatically, under support of Open AI.

An alternative solution would be to get the comments allocated to the columns manually, e.g., by the Scrum Master facilitating the retro meeting, but this would slow down the meeting. In our prototype we use ChatGPT API to provide a preliminary solution for this task.

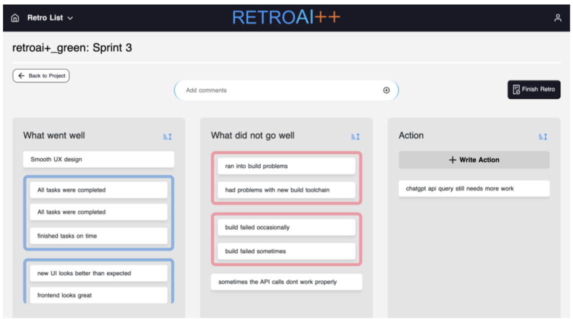


coffee4U_green: Sprint 4

good retro

Figure 3: RetroAI++ retro board: General structure (light version of the UI).

Often, team members submit comments that are semantically similar: if something was very good or upsetting during a sprint, it's very likely that many team members will have the same feelings about it. It makes sense to visually group similar comments as this decreases the cognitive load of the board analysis. We implemented this idea by presenting similar comments within a group highlighted with a colour frame: blue for *"What went well"* and red for *"What didn't go well"*, see Figure 4.



retroai+_green: Sprint 3

Smooth UX design

All tasks were completed

All tasks were completed

finished tasks on time

new UI looks better than expected

frontend looks great

ran into build problems

had problems with new build toolchain

build failed occasionally

build failed sometimes

sometimes the API calls dont work properly

chatgpt api query still needs more work

Figure 4: RetroAI++ retro board: grouping similar comments.

In the current version of the prototype, grouping has been implemented as a manual functionality (which can be applied by any team member), but as the future work we would like to explorer application of AI approaches to allow the team have this boring but important task done for them automatically. As preliminary solution, we provide a functionality to sort comments by the frequency of their appearance, which might streamline the currently manual process.

RetroAI++ can also provide a summary of a sprint, generated using ChatGPT based on a Kanban board for this sprint. This information can serve as a starting point for a retro-meeting, as the autogenerated summary provides a short overview of the team's progress over the sprint wrt. to the Sprint backlog, i.e., wrt. the plan the team had for this sprint.

7 THREATS TO VALIDITY

There are several threats to validity of our experiments. The first threat is the limited scope of the benchmark dataset, which was limited to 200 retro-comments and created manually, which means it obviously doesn't cover fully the infinite set of all possible retro-comments that can be written in the real life meetings. However, our manually created benchmark dataset has a significant size and covers typical points that arise in the retrospectives in real industrial projects.

The second threat is the limited number of runs for each prompt presented in the experiment analysis in Sections 4 and 5. While running our experiments, we observed that the results of prompt executions might differ slightly, i.e., if we run the same prompt multiple times the responses of ChatGPT might be not exactly the same. However, we haven't observed any statistically significant difference, therefore due to space limit we restrict only discussion to the analysis of a single run per each prompt.

Also, the dataset used in our experiment has been created and manually labelled by the second author and then refined and extended by the first author, based on the experience from industrial projects. The analysis and classification of the ChatGPT responses has been done manually by the authors. To mitigate the issues with incorrect labelling and classification, we used peer-reviewing strategy.

8 CONCLUSIONS

In this paper, we presented our ongoing research on streamlining Agile/Scrum processes with the support

of AI approaches. We discussed our experiments with OpenAI's ChatGPT-4 turbo to analyse the applicability of generative AI for supporting Agile/Scrum retrospective meetings and summarised the core lessons learned from these experiments. We also presented our prototype tool RetroAI++, whose aim is to automate and simplify Agile/Scrum processes for software development projects. We especially focused on RetroAI++ functionality to facilitate retro-meetings.
Future work: As our future work, we plan to conduct experiments on a larger dataset and to refine/extend our prototype.

ACKNOWLEDGEMENTS

We would like to thank Shine Solutions for sponsoring this project under the research grant PRJ00002505, and especially Branko Minic and Adrian Zielonka for priding their industry-based expertise and advices. We also would like to thank students who contributed to creation of earlier versions of the RetroAI tool: Weimin Su, Ahilya Sinha, Hibbaan Nawaz, Kartik Kumar, Muskan Aggarwal, Justin John, Shalvi Tembe, Niyati Gulumkar, Vincent Tso, and Nguyen Duc Minh Tam.

REFERENCES

Al-Saqqa, S., Sawalha, S., and AbdelNabi, H. (2020). Agile software development: Methodologies and trends. *Int. Journal of Interactive Mobile Technologies*, 14(11).

digital/ai (2023). 17th state of agile report.

Duehr, K., Efremov, P., Heimicke, J., Teitz, E. M., Ort, F., Weissenberger-Eibl, M., and Albers, A. (2021). The positive impact of agile retrospectives on the collaboration of distributed development teams–a practical approach on the example of Bosch Engineering GMBH. *Design Society*, 1:3071–3080.

Erdoğan, O., Pekkaya, M. E., and Gök, H. (2018). More effective sprint retrospective with statistical analysis. *Journal of Software: Evolution and Process*, 30(5).

Fernandes, S., Dinis-Carvalho, J., and Ferreira-Oliveira, A. T. (2021). Improving the performance of student teams in project-based learning with Scrum. *Education sciences*, 11(8):444.

Gaikwad, P. K., Jayakumar, C. T., Tilve, E., Bohra, N., Yu, W., and Spichkova, M. (2019). Voice-activated solutions for agile retrospective sessions. *Procedia Computer Science*, 159:2414–2423.

Hakim, H., Sellami, A., and Ben-Abdallah, H. (2024). MPED-SCRUM: An automated decision-making framework based measurement for managing requirement change within the Scrum process. In *ENASE*, pages 571–581.

Jovanović, M., Mesquida, A.-L., Radaković, N., and Mas, A. (2016). Agile retrospective games for different team development phases. *Journal of Universal Computer Science*, 22(12):1489–1508.

Kadenic, M. D., Koumaditis, K., and Junker-Jensen, L. (2023). Mastering Scrum with a focus on team maturity and key components of Scrum. *Information and Software Technology*, 153:107079.

Khanna, D. and Wang, X. (2022). Are your online agile retrospectives psychologically safe? the usage of online tools. In *XP'22*, pages 35–51. Springer.

Marsden, N., Ahmadi, M., Wulf, V., and Holtzblatt, K. (2021). Surfacing challenges in scrum for women in tech. *IEEE Software*, 39(6):80–87.

Marshburn, D. (2018). Scrum retrospectives: Measuring and improving effectiveness. In *SAIS*.

Matthies, C. (2020). Playing with your project data in scrum retrospectives. In *ACM/IEEE 42nd International Conference on Software Engineering*, ICSE '20, page 113–115. ACM.

Matthies, C. and Dobrigkeit, F. (2021). Experience vs data: A case for more data-informed retrospective activities. In *XP'21*, pages 130–144. Springer.

Mich, D. and Ng, Y. Y. (2020). Retrospective games in Intel Technology Poland. In *FedCSIS*, pages 705–708. IEEE.

Ng, Y. Y. and Kuduk, R. (2024). Implementing action items over improving the format of retros. In *SAC*, pages 853–855.

Ng, Y. Y., Skrodzki, J., and Wawryk, M. (2020). Playing the sprint retrospective: a replication study. In *Advances in Agile and User-Centred Software Engineering*, pages 133–141. Springer.

Przybyłek, A., Albecka, M., Springer, O., and Kowalski, W. (2022). Game-based sprint retrospectives: multiple action research. *Empirical Software Engineering*, 27(1):1.

Przybyłek, A. and Kotecka, D. (2017). Making agile retrospectives more awesome. In *FedCSIS*, pages 1211–1216. IEEE.

Schwaber, K. and Sutherland, J. (2011). The Scrum guide. *Scrum Alliance*, 21(1):1–38.

Spichkova, M. (2019). Industry-oriented project-based learning of software engineering. In *Int. conference on engineering of complex computer systems (ICECCS)*, pages 51–60. IEEE.

Sun, C., Zhang, J., Liu, C., King, B. C. B., Zhang, Y., Galle, M., Spichkova, M., and Simic, M. (2019). Software development for autonomous and social robotics systems. In *Intelligent Interactive Multimedia Systems and Services*, pages 151–160. Springer.

Torchiano, M., Vetrò, A., and Coppola, R. (2024). Teaching scrum with a focus on efficiency and inclusiveness. In *EASE*, pages 595–599.

White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., Elnashar, A., Spencer-Smith, J., and Schmidt, D. C. (2023). A prompt pattern catalog to enhance prompt engineering with ChatGPT. *arXiv preprint arXiv:2302.11382*.