PACGBI: A Pipeline for Automated Code Generation from Backlog Items

Mahja Sarschar
Hochschule für Technik und Wirtschaft Berlin
Germany
Mahja.S@web.de

Gefei Zhang
Hochschule für Technik und Wirtschaft Berlin
Germany
gefei.zhang@htw-berlin.de

Annika Nowak
Capgemini Deutschland GmbH
Germany
annika.nowak.public@gmail.com

Abstract

While there exist several tools to leverage Large Language Models (LLMs) for code generation, their capabilities are limited to the source code editor and are disconnected from the overall software development process. These tools typically generate standalone code snippets that still require manual integration into the codebase. There is still a lack of integrated solutions that seamlessly automate the entire development cycle, from backlog items to code generation and merge requests. We present the Pipeline for Automated Code Generation from Backlog Items (PACGBI), an LLM-assisted pipeline integrated into GitLab CI. PACGBI reads backlog items in the code repository, automatically generates the corresponding code, and creates merge requests for the generated changes. Our case study demonstrates the potential of PACGBI in automating agile software development processes, allowing parallelization of development and reduction of development costs. PACGBI can be utilized by software developers and enables non-technical stakeholders and designers by providing a holistic solution for using LLMs in software development. A screencast of this tool is available at https://youtu.be/TI53m-fIoyc, its source code at https://github.com/Masa-99/pacgbi.

ACM Reference Format:
Mahja Sarschar, Gefei Zhang, and Annika Nowak. 2024. PACGBI: A Pipeline for Automated Code Generation from Backlog Items. In *39th IEEE/ACM International Conference on Automated Software Engineering (ASE '24), October 27-November 1, 2024, Sacramento, CA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3691620.3695346

1 Introduction

Modern software development is deeply coupled with version control systems like Git and Web platforms supporting them like GitHub or GitLab [8]. Besides the source code, requirements for the improvement of the software under development, or *backlog items* in the terminology of agile development, are also stored on such platforms. The workflow of implementing a backlog item usually starts with creating a new *branch*, which we call the *item branch*, then code for the item is put into the item branch. When coding is finished, a *merge request* is created and can be reviewed by the development team. When accepted, the item branch is merged back into the main branch.

Code generation using Large Language Models (LLM) has been recognized as promising. Several academic and commercial tools [5, 14] show how LLM-assistance can help to code more efficiently. However, it is only code generation that is supported by these tools, and the generation is done separately from the other steps. In order to really harness the power of LLMs for the software development industry, it is desirable to integrate LLMs into the complete workflow rather than solely using them for code generation.

To this end, we developed the Pipeline for Automated Code Generation from Backlog Items (PACGBI). PACGBI provides a holistic support from reading a backlog item over code generation to the generation of a merge request. This way, PACGBI helps enhance the efficiency of software development by integrating LLM-assistance in GitLab.

2 Related Work

LLMs are already used in multiple different tools to enhance the coding experience. There are in-IDE tools like GitHub Copilot, Amazon CodeWhisperer or TabNine [5, 14]. They are installed as a plugin in source-code editors and deliver code completion suggestions based on the current code file within the context of the repository code. For web development, there are tools that can specifically generate user interfaces (UI). Anima[1] for example transfers mockups into React code, while v0[2] by Vercel turns natural language into React components via Few-Shot prompting.

Despite their usefulness [13, 15], the power of these tools is still limited to code generation only, separated from the other steps of the software development. Jimenez et al. [6] investigate large language models to resolve bugs or to implement new features from Github using SWE bench evaluation. Within a 3-stage-pipeline pull requests are created that perform the needed changes. Their pipeline is limited to Python task instances which need to pass tests. In comparison, our tool in the form of a CI-pipeline provides a holistic approach which is fully integrated into the usual software development process, in this case GitLab, and focus on web development tasks in React. It starts from the code repository hosting platforms and automates the development of tasks with the power of LLMs. Furthermore, GitHub Copilot Workspace [4] and Codegen[3] are expected to be released in 2024.

ASE '24, October 27-November 1, 2024, Sacramento, CA, USA
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-1248-7/24/10
https://doi.org/10.1145/3691620.3695346

[1]https://www.animaapp.com/, accessed on 2024-06-25.
[2]https://v0.dev/, accessed on 2024-06-25.
[3]https://www.codegen.com/, accessed on 2024-06-26.

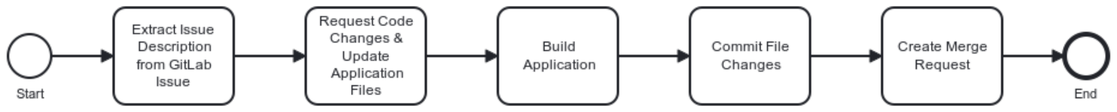
Start
Extract Issue Description from GitLab Issue
Request Code Changes & Update Application Files
Build Application
Commit File Changes
Create Merge Request
End

Figure 1: Architecture of PACGBI [10]

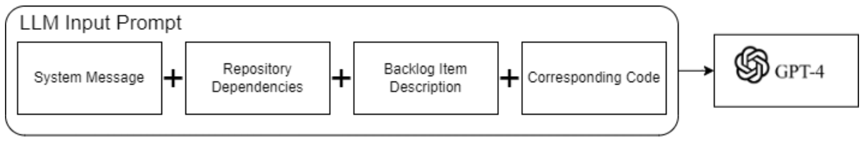

Figure 2: Structure of the Input Prompt provided to the LLM [10]

3 PACGBI

PACGBI integrates with the user's code repository as a pipeline through GitLab's Continous Integration (CI). Generally speaking, pipelines are automated sequences designed to perform software development tasks like building, testing, and code analysis. A pipeline consists of *jobs* and *stages*. The former defines the tasks which are done in a certain sequence, and stages group jobs and define when they are run. [1] PACGBI consists of five stages, as shown in Fig. 1, which reflect the process that developers go through during their daily development tasks.

In the first pipeline stage, the issue description and label are fetched from the backlog item in GitLab, from which PACGBI was triggered. Currently, the pipeline is implemented to modify only one file in the underlying repository, an extension to facilitate changing multiple files is planned. Next, PACGBI prompts the LLM to generate new code with the input prompt shown in Fig. 2 and explained in the next subsection. The response of the LLM is the regenerated code file including the requirements. Then the application is built, and if that is successful, the changes are committed in the fourth stage. Finally, PACGBI creates a merge request to make the changes reviewable for the development team.

3.1 Prompt Structure

In order to generate code with LLM, an input prompt must be provided. LLMs have achieved good results with the intuitive zero shot promping method where the task is inputted directly to the LLM [7]. PACGBI constructs the input prompt from four components, as shown in Fig. 2.

The first component is the following static system message: *"As a senior developer on a web development team, you are responsible for developing a payment front end application. It is important to regenerate the entire code file each time, without providing explanations for the code."* It first defines a persona for the LLM to act as, and thus provides a context for the generation of the desired output [11, 13]. Then it gives instructions to regenerate a whole code file, and not to explain the code. These instructions define how the model should perform its task and by avoiding explanations, the token usage can be kept low.

The second component comprises the dependencies of the source code. They are included to provide more context about the used libraries, frameworks and their versions, aiming to improve the correctness of the generated code. The third component contains the backlog item description, which serves as key information for the code generation. Lastly, the corresponding code file to be enhanced by the LLM to implement the backlog items' requirements is added to the prompt. It is retrieved by using the issue label of the backlog item to find the corresponding repository file.

These four components, in addition to two parameters specifying the LLM model to use and the temperature, are built into one request to the LLM. As stated before, we use the model GPT-4-Turbo (*gpt-4-0125-preview*) and the temperature is set to 0, as a lower temperature is more suitable for single output generation [2, 12].

3.2 Usage

Generally, PACGBI is designed to be used in any code repository, independent of the programming language, development environment and application use case. Currently, it is built for GitLab CI, using OpenAI's Chat API for code generation and Prettier[4] to enforce repository-specific code formatting rules on the LLM-generated code.

To use PACGBI, a GitLab Runner is needed, and the backlog item must be labeled with the name of the code file which should be modified. Then, the user can trigger PACGBI by creating a branch in the repository with the prefix `bot/` and the backlog item id. The process starts automatically and depending on the size of the backlog item and the application, the merge request of the implementation of PACGBI is created within a few minutes.

Our envisioned users are software developers who want to automate parts of their daily development tasks. Also, (non-)technical stakeholders like product owners are enabled to realize their backlog items automatically for faster development.

Further, after using PACGBI, data pairs consisting of natural language inputs in form of backlog items and their corresponding code outputs are available. These data can serve as high-quality training and fine-tuning datasets for LLMs developers.

4 Evaluation

We evaluated PACGBI by applying it on eight backlog items[5] of a React-based front end application. In advance, a Scrum team

[4]https://prettier.io/, accessed on 2024-06-08.
[5]For the case study, we chose existing backlog items which only needed modification in one file.

Issue ID	Summary	Complexity	Overall Rating (from 0 to 5)			Review Decision	Development Time
			CQ	F	UI		
GenAI-MS-001	Rename "New" Button and add Tooltip	1	5	4	5	Accepted	00:09:10
GenAI-MS-002	Add Date Input for Transactions on new Transactions Dialogue	1.4	1	2	2	Requested Rework	00:09:49
GenAI-MS-003	Add Transaction Status to Overview List	2	2	2	1	Requested Rework	00:07:41
GenAI-MS-005	Add Visualization of Transaction Status	4	0	0	0	Rejected	00:08:17
GenAI-MS-006	Additional Information added to the Account Balance	5	0	0	0	Rejected	00:07:11
GenAI-MS-008	Add "View"-button to Notifications	1.4	5	4	5	Accepted	00:07:30
GenAI-MS-009	Add a Comment Section in the List Overview of Transactions	2.6	3	5	5	Accepted	00:11:02
GenAI-MS-010	Make Interaction Possibility visible for Items on the Transaction List	1	4	4	4	Requested Rework	00:07:38

Table 1: Overview of the evaluation of PACGBI.

estimated their complexity in user story points with the Planning Poker method. For all backlog items, PACGBI successfully generated code, pushed it to the repository, and created merge requests.

Without modifying the merge requests, we assessed code validity, quality, development time, token usage, and costs automatically and three senior developers performed manual code reviews. An overview of the results is shown in Table 1.

4.1 Code Validity

Code validity, defined as compliance with programming language syntax rules [15], was assessed through automatic builds using the package manager yarn [6] in PACGBI. If the project can be built successfully, then the code is syntactically correct.

In our test, all merge request (8/8) passed the build-job successfully without errors, affirming the syntactical validity of the generated code.

4.2 Code Quality

We use the tool SonarCloud[7] to measure the quality of the generated code. Our assessment is based on SonarCloud's default *quality gate*, which is called *Sonar way*, but the criterion "test coverage" is excluded. Sonar way evaluates the maintainability, reliability, and security on a scale from A to E, A being the best. This check is performed automatically by PACGBI.

The generated code is considered high quality when less than 30% of lines are duplicated, it has maintainability, security and reliability ratings of A and all security hotspots are 100% reviewed.

All merge requests (8/8) have passed the requirements of the quality gate. Only one merge request shows two issues, which are both related to maintainability and rated low impact by SonarCloud. These issues were caused by unused imports in a TypeScript file modified by PACGBI.

[6]https://yarnpkg.com/, accessed on 2024-03-26.
[7]https://sonarcloud.io/, accessed on 2024-06-25.

4.3 Development Time

We refer to the time from triggering PACGBI for a backlog item to the created merge request as its *development time*.

In average, the pipeline required eight minutes and 32 seconds to complete the entire process. The duration varied between tasks, with code generation typically ranging from 57 seconds to 3 minutes, building the application spanning around 3 minutes, and the total pipeline duration fluctuating between 7 minutes and 11 seconds to 11 minutes maximum.

4.4 Token Usage And Costs

In average, the input prompt is translated into 3097 tokens and 838 tokens are generated for the output of 140 lines of code. According to the current pricing model the generation of the eight backlog items by PACGBI costs approximately a total of $0.45.

4.5 Manual Code Review

To stick to the process of software development in the industry and gain practical insights about the code quality of LLM-generated code, we asked three senior developers from the company Capgemini to conduct code reviews for the merge requests created by PACGBI. They were also requested to provide an overall rating of the generated code in a 5-point scale with respect to (1) code quality, (2) functionality and (3) quality of the UI. Lastly, the senior developers were asked to state whether they would accept, reject or request rework for the merge request using the principles defined in [3]. Then, we conducted the following analysis:

(1) We calculated the number of comments made in the reviews per 100 lines of code.
(2) We analysed the reviews' texts and conducted a qualitative content analysis based on [9].
(3) We analysed the worst and best rated criteria (code quality, functionality and quality of UI) and used Spearman-Rho rank correlation to test a correlation between overall rating and issue complexity.

(4) We calculated the success of PACGBI by counting how many times it generated merge requests which were accepted by the senior developers.

In average, 2.81 comments per 100 lines of code were made by the senior developers, ranging from 0.59 to 12.24.

4.5.1 Review Comments The code reviews' text were analysed through two reduction steps according to [9], resulting in six distinct categories of code review topics:

Category ID	Category Name	Occurences
C1	Code Formatting	10
C2	Functionality Defect	6
C3	UI Styling	2
C4	TypeScript Error	2
C5	Accessibility Issue	1
C6	Unused Code	1

Most of the comments are categorized as "C1 Code Formatting". In six out of ten cases, the problem is a missing line at the end of the file. Three comments are about double quotation marks being used instead of single ones and code being formatted in multiple lines instead of one. This kind of comments can be considered a manual report of the finding of the automatically executed tool Prettier, which is used in the repository to automatically report formatting issues. And one comment is just generally *"'prettier' issues in all modified lines of code"*.

The category "C2 Functionality Defect" was reported six times in three different merge requests. Some examples are: *"no possibility to enter date manually. Dialog is always open. I'd expect to open the dialog on click or manually type a date"* or *"multiple errors. Doesn't work. Setting of transactionStatus doesn't work properly."* "C3 UI Styling" was mentioned two times. One example is the comment *"UI inappropriate. There is only some text next to the app header. Also information disappears on mobile => lacking responsiveness"*. "C4 TypesScript Errors" was reported two times in two different merge requests. One is about a type warning, another is about a TypeScript error with a component of the `material-ui` library used to implement the UI. Also, "C5 Accessibility Issues" and "C6 Unused Code" were reported once each.

4.5.2 Overall Rating As shown in Table 1, the average rating of UI is 2.5, code quality (CQ) is 2.6 and functionality (F) is 2.8. Therefore, functionality was rated the best, while UI was rated the worst.

The results of the Spearman-Rho correleation test show that only the UI rating is significantly correlated with the complexity of the backlog item. The p-value is 0.019 and is therefore lower as the significance level of 0.05. For functionality, it is 0.083, and for code quality, no correlation could be proven. All of this correlation coefficients are negative which means when the complexity is high, the overall ratings are low.

Among the overall ratings, a strong correlation could be determined between functionality and code quality as well as between functionality and UI (p-value <= 0.01).

4.5.3 Review Decision Regarding the review decision, three merge requests with AI-generated code have been accepted, for three merge requests a rework was required and two were rejected, as shown in Table 1. This results in an acceptance rate of 37.5%.

5 Conclusions and Future Work

We have presented PACGBI, an LLM-based CI-pipeline that uses OpenAI to generate code from backlog items, builds the application and creates a merge request. Our case study shows that the holistic approach of PACGBI promises better efficiency, lower development costs and duration for the development of React applications.

For future work, we plan to extend PACGBI's capabilities to support multiple file modifications and different LLMs. Also, we want to perform more evaluations, especially on complex task and with different nuances of backlog item descriptions. Further, the underlying concept can be implemented in other code hosting platforms with CI integration, like GitHub. Last but not least, leveraging the data generated by PACGBI for further training and fine-tuning of LLMs could significantly improve the performance and applicability of PACGBI in real-world software development settings.

References

[1] S.A.I.B.S. Arachchi and Indika Perera. 2018. Continuous Integration and Continuous Delivery Pipeline Automation for Agile Software Project Management. In *2018 Moratuwa Engineering Research Conf. (MERCon)*. 156–161.

[2] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. 2021. Program Synthesis with Large Language Models. *ArXiv* abs/2108.07732 (2021). https://api.semanticscholar.org/CorpusID:237142385

[3] Nicole Davila and Ingrid Nunes. 2021. A Systematic Literature Review and Taxonomy of Modern Code Review. *J. Syst. Softw.* 177 (2021), 110951.

[4] Thomas Dohmke. 2024. GitHub Copilot Workspace: Welcome to the Copilot-native developer environment - The GitHub Blog. https://github.blog/2024-04-29-github-copilot-workspace/, accessed on 2024-06-27.

[5] Yujia Fu, Peng Liang, Amjed Tahir, Zengyang Li, Mojtaba Shahin, and Jiaxin Yu. 2023. Security Weaknesses of Copilot Generated Code in GitHub. *CoRR* abs/2310.02059 (2023). https://doi.org/10.48550/ARXIV.2310.02059 arXiv:2310.02059

[6] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. SWE-bench: Can Language Models Resolve Real-world Github Issues?. In *12th Int. Conf. Learning Representations*. https://openreview.net/forum?id=VTF8yNQM66

[7] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large Language Models are Zero-Shot Reasoners. *ArXiv* abs/2205.11916 (2022). https://api.semanticscholar.org/CorpusID:249017743

[8] Rana Majumdar, Rachna Jain, Shivam Barthwal, and Chetna Choudhary. 2017. Source code management using version control system. In *2017 6th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*. 278–281. https://doi.org/10.1109/ICRITO.2017.8342438

[9] Philipp Mayring. 2014. *Qualitative Content Analysis: Theoretical Foundation, Basic Procedures and Software Solution.* Social Science Open Access Repository (SSOAR), Klagenfurt. 143 pages. https://nbn-resolving.org/urn:nbn:de:0168-ssoar-395173, accessed on 2024-05-20..

[10] Mahja Sarschar. 2025. *Pipeline for Automated Code Generation from Backlog Items (PACGBI)—Analysis of Potentials and Limitations of Generative AI for Web Development.* Springer Nature. To appear..

[11] Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C. Schmidt. 2023. A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT. arXiv:2302.11382 [cs.SE]

[12] Frank F. Xu, Uri Alon, Graham Neubig, and Vincent Josua Hellendoorn. 2022. A Systematic Evaluation of Large Language Models of Code. In *Proc. 6th ACM SIGPLAN Int. Symp. Machine Programming.* ACM, 1–10.

[13] Dapeng Yan, Zhipeng Gao, and Zhiming Liu. 2023. A Closer Look at Different Difficulty Levels Code Generation Abilities of ChatGPT. In *Proc. 38th IEEE/ACM Int. Conf. Automated Software Engineering, (ASE'23)*. IEEE, 1887–1898.

[14] Burak Yetistiren, Isik Özsoy, Miray Ayerdem, and Eray Tüzün. 2023. Evaluating the Code Quality of AI-Assisted Code Generation Tools: An Empirical Study on GitHub Copilot, Amazon CodeWhisperer, and ChatGPT. *ArXiv* abs/2304.10778 (2023). https://api.semanticscholar.org/CorpusID:258291698

[15] Burak Yetistiren, Isik Ozsoy, and Eray Tuzun. 2022. Assessing the Quality of GitHub Copilot's Code Generation. In *Proc. 18th Int. Conf. Predictive Models and Data Analytics in Software Engineering, PROMISE'22*, Shane McIntosh, Weiyi Shang, and Gema Rodríguez-Pérez (Eds.). ACM, 62–71.