ADA-Gen: Iterative and Incremental Generation of Full-Stack Apps for Learning Agile/DevOps Software Development Practices

Ta Nguyen Binh Duong[a]
School of Computing and Information Systems, Singapore Management University, Singapore

Keywords: Full-Stack App Generation, Teaching and Learning Agile/DevOps Practices, Iterative and Incremental Software Development, Large Language Models.

Abstract: To learn Agile/DevOps practices effectively, students need to apply them in an actual software development project. This is challenging if students are mostly from non-computing backgrounds and they do not have time in the curriculum to learn programming and related tools. Therefore, it is important to help students who do not possess programming foundations to develop fully functional software during the process of learning Agile/DevOps concepts. We noted that existing low-code/no-code app development platforms have not been designed to teach Agile/DevOps practices. On the other hand, recent AI-based tools for code generation such as GitHub Copilot have been built mainly for programmers. In this work, we designed and implemented ADA-Gen (Agile/DevOps App Generator), a teaching tool leveraging large language models (LLMs) to generate full-stack web apps following an iterative and incremental development methodology widely practiced in Agile/DevOps circles. ADA-Gen is integrated with Jira, one of the most popular platforms for software project management, so students can use it right away without much further setup. This approach allows non-computing students to experience the complete full-stack software development life cycle. We have conducted extensive evaluations of ADA-Gen using various realistic project scenarios. The evaluations demonstrated the capabilities of ADA-Gen in full-stack web app generation, and in providing plenty of learning opportunities for students to appreciate key Agile/DevOps practices.

1 INTRODUCTION

Agile and DevOps practices have been widely used in software development to enhance team productivity and software quality (Mehta and Sood, 2023). Common Agile methodologies (Schwaber and Sutherland, 2011) are characterized by iterative and incremental development of working software, frequent feedback, and adaptability to changing requirements. They have been shown to increase the success rates of software projects (Palopak and Huang, 2024). On the other hand, DevOps (Amaro et al., 2022) extends the Agile philosophy by leveraging automation to bridge the gap between development and operations teams; enabling frequent releases of production software and increased customer satisfaction.

Although there have been many variations of Agile (Al-Baik et al., 2024), in popular methods such as Scrum (Schwaber and Sutherland, 2011), the project is broken down into iterative and incremental cycles called sprints, which typically last around 2-4 weeks. The software requirements/features are usually detailed in the form of user stories or epics (Cohn, 2004), which are descriptions of functionality from the user's perspective. In each sprint, several selected user stories can be further broken down into child tasks (or sub-tasks) and implemented. A working software prototype must be demonstrated by the end of the sprint to the stakeholders. Agile's emphasis on iterative/incremental development and frequent release of working software can be supported by DevOps practices (Amaro et al., 2022) such as continuous integration (CI), which is basically the frequent, automated integration and testing of small code changes from team members in a shared repository.

Agile/DevOps practices have been covered widely in software engineering courses at higher education institutions. It has been noted that Agile teaching should emphasize hands-on experience rather than theoretical knowledge of specific Agile methods (Devedžić et al., 2010; Omidvarkarjan et al., 2023). Other studies have also shown that project-based learning approaches, where students engage in

[a] https://orcid.org/0000-0002-2882-2837

Ta, D. N. B.
ADA-Gen: Iterative and Incremental Generation of Full-Stack Apps for Learning Agile/DevOps Software Development Practices.
DOI: 10.5220/0013422800003932
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 17th International Conference on Computer Supported Education (CSEDU 2025) - Volume 2*, pages 363-370
ISBN: 978-989-758-746-7; ISSN: 2184-5026
Proceedings Copyright © 2025 by SCITEPRESS – Science and Technology Publications, Lda.

actual software development, lead to improved understanding of Agile principles (Almeida, 2012). However, it is challenging for non-computing students (e.g., in short courses for post-graduate certificates) to obtain such hands-on experience as they are not able to actually implement and demonstrate working software in the short time frame of a semester. Low-code or no-code development environments, e.g., Microsoft Power Apps, have been used to teach Agile processes (Lebens and Finnegan, 2021). We note that such approach might not be able to provide a realistic hands-on development experience.

Recently, pre-trained large language models (LLMs) such as the GPT based models have been widely used in software development tasks such as code completion, test case generation, code review, etc. (Ozkaya, 2023). Existing tools like OpenDevin (Wang et al., 2024), GitHub Copilot, Amazon CodeWhisperer (Li et al., 2024) have been introduced to improve software development productivity. However they are designed primarily for software engineers rather than non-computing students without background in programming. More recent AI-based tools (Brockenbrough and Salinas, 2024; da Silva Neo et al., 2024) leverage LLMs to generate or improve user stories, but this is just one aspect of Agile/DevOps practices. To the best of our knowledge, there has been no LLM-based code generation tool designed specifically for helping non-computing students learn Agile/DevOps concepts more effectively. This is the focus of our work.

2 ADA-GEN

2.1 Objectives

In this work, the goal is to assist students with no or very little programming background to learn Agile/DevOps practices by experiencing a real-world, typical software development life cycle. ADA-Gen is not designed to generate very complex apps used in critical business processes. This approach differs from existing frameworks like OpenDevin (Wang et al., 2024), which focus on more advanced application development and designed primarily for software engineers. However, it is important that our tool provides exposure to common technical issues, e.g., bug fixing, requirement improvement, testing, integration, etc., in real-world Agile/DevOps projects. To this end, we consider the following key design objectives of ADA-Gen:

- Generating code from popular project management interfaces such as Jira. This is to help students appreciate Agile practices without leaving their familiar tools and environments.
- Providing learning opportunities for a deeper appreciation of Agile/DevOps practices such as user story writing, task breakdown, task estimation, velocity, continuous integration, etc. This is because learning practices and theoretical processes using only planning tools like Jira without experiencing real development is not sufficient (Omidvarkarjan et al., 2023).
- Supporting iterative and incremental development. The code generation process should be carried out per sprint. In addition, subsequent sprints should incorporate code generated in earlier sprints for continuity and consistency. In this way, students can appreciate the importance of making continuous and incremental improvements and adjustments based on feedback during each sprint review.
- Generating ready-to-run, full-stack web apps with both front-end and back-end components. This is because students will need to do a demonstration to stakeholders at the end of each sprint.

2.2 System Design

ADA-Gen integrates a software project management platform (Jira), a hosted Git repository for code version control (Bitbucket), and a pre-trained LLM (OpenAI's GPT-4o) into a learning tool that non-computing students can use to learn and experience incremental and iterative development practices used in Agile/DevOps processes. Our design, as shown in Figure 1, is based on a microservices architecture which includes the following components:

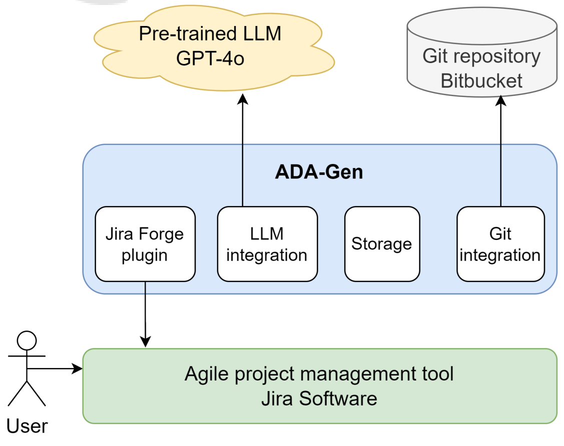

Figure 1: Components in ADA-Gen's software architecture.

Jira Forge Plugin. A Jira software plugin developed using Atlassian Forge[1], which is a cloud app development platform provided by Atlassian. Apps built with Forge are hosted entirely on Atlassian serverless cloud infrastructure. This enables the creation of custom Jira apps integrated directly into the Jira user interface. As most students should already be familiar with the use of Jira for project management, the learning curve is reduced for ADA-Gen. The plugin is implemented in JavaScript and deployed on Atlassian cloud. The Forge SDK allows the plugin to make calls to the Jira APIs to fetch stories and other project related information for code generation.
LLM Integration. A Java Spring Boot service which can be hosted on any public clouds or locally. The service provides integration with a selected LLM deployment, e.g., OpenAI. In our implementation, we use OpenAI's GPT-4o APIs. This Spring Boot service also handles requests from the Jira pages and communicates with the database storage.
Git Integration. ADA-Gen requires the integration of a hosted Git repository such as GitHub or Bitbucket. This is essential for maintaining the generated source code and configuration files for sprint by sprint code generation, and continuous integration.
Storage. ADA-Gen has a storage component which uses a relational database for important project-related information such as LLM API keys and Git repository access tokens.

2.3 Prompt Engineering

The main challenge in implementing ADA-Gen is to generate incremental code in multiple consecutive sprints while maintaining consistency and ensuring that a proper working prototype can be created after a few sprints. ADA-Gen makes use of LLMs, in particular OpenAI's GPT-4o. A prompt is constructed with the required user story and sent to the LLM, which will reply with generated code (HTML, CSS, JavaScript, and Python) implementing the story. Constructing appropriate LLM prompts for a particular application domain is referred to as prompt engineering, which is a currently active research area (Oppenlaender et al., 2024).

In ADA-Gen, we have devised a prompting strategy with multiple stages. The prompt provides information not just about the current story to be generated, but also the relevant context from sibling and/or child tasks, desired repository structure and existing code, templates for expected code structure, generation guidelines, and directives for potential bug prevention. The below details are included in the prompt sent to the LLM for code generation. Multiple stories in the same sprint can be generated sequentially in this way.

[1]https://developer.atlassian.com/platform/forge

Initial Setup. At the start of the prompt, we provide instructions to set the overall guidelines for the LLM, ensuring that generated code focuses on the main user story and its child tasks while considering related stories for context. The following instructions are used in the prompt:

```
Generate code strictly for the
    following user story and its child
    tasks. You will be provided with
    sibling user stories and parent
    epics as additional context to
    assist in generating code for the
    user story. You will also receive a
     list of relevant files existed in
    the repository to ensure the
    generated files do not overlap with
     those already present. Make
    necessary changes to these existing
     files to fulfill the specific
    requirements of the user story and
    its child tasks, but do not remove
    or alter any existing functionality
     that serves other user stories.
```

Prompt Structure Information. In the next stage, we provide a layout to help the LLM understand the prompt structure and context of various tasks. This could make it easier to generate more relevant and coherent responses. Below is the prompt structure given to the LLM:

```
User story and its child tasks
Parent epic and sibling stories if any
Relevant files in repository if any
Code templates for back-end and front-
    end components
Generation guidelines for code
Bug prevention directives
```

User Story Details. In this stage, we provide details for the user story for which code has to be generated by the LLM. The user story's title, ID, and description are automatically retrieved from the Jira project via the Forge plugin when user chooses to do code generation. We specify the story as follows:

```
<User story to generate code for>
    Title, ID, and description of the
    user story
</User story to generate code for>
```

Child Tasks, Parent Epics and Sibling Stories. In this part of the prompt, we provide information about child tasks, related sibling stories and parent epics to provide the appropriate context for the LLM to do code generation. The LLM is not supposed to generate code for the parent and sibling stories.

Repository Integration. This step is to provide the LLM with context on existing code and configuration files in the Git repository of the project. This helps in understanding what has already been generated to avoid duplicated code and ensure consistency and continuity in the code.
Code Templates. This stage provides templates to guide the LLM on how to structure the generated code for components of the application such as back-end, front-end, testing, Docker files, and Bitbucket pipeline configuration for continuous integration. The prompt includes several sample code files to provide a template for back-end code that should be generated. Note that we do not show the complete code here due to space limitation.
Generation Guidelines. This part specifies the styles and standards for the generated code, such as code review, common code smells to avoid, e.g., lengthy code, etc. Examples includes: 1) if the script gets too long, divide it into multiple scripts, 2) after generating the code, it is crucial to perform a thorough review and validation of the entire script, etc.
Bug Prevention Directives. List potential bugs that frequently occur and may not be caught without explicit instructions during code generation. This helps steer the LLM away from known issues, such as incorrect targets in configuration files, or deprecated code that might be generated. For example, some Flask libraries used by the LLM during code generation such as `before_first_request`, have already been deprecated at the time of our experiment. This leads to compilation issues for the generated code.

3 EVALUATION

In the evaluation, we aimed to answer the following research questions (RQs):

- **RQ1.** It is possible for those without programming knowledge to use ADA-Gen to create a complete, full-stack web app following an iterative and incremental development process in Agile/DevOps?
- **RQ2.** Does ADA-Gen provide learning opportunities for students to identify, implement and appreciate important Agile/DevOps practices in building actual software?

To answer these RQs, we designed several scenarios which usually occur when students run software development projects, based on our experience teaching a large number of non-computing students the Agile/DevOps methodology over the years at our university. The students in our Agile/DevOps courses were post-graduates with zero, or very little background in programming. The project scenarios range from a simple setup in which all stories are completed in one sprint, to those with multiple sprints and a complex story structure incorporating real-world application features.

For both RQs, we played the role of a small student team without any programming background. To ensure a realistic evaluation, we did not attempt to fix any issues in the generated code manually, instead we just relied on ADA-Gen to do the generation and bug fixing. For **RQ1**, we looked at the generated code and considered the bugs with regard to the requirements defined in the stories, and compilation/runtime errors encountered after the app has been generated in a sprint. After all issues were fixed, we ran the generated app and evaluated if it meets the desired requirements.

For **RQ2**, while using ADA-Gen, we identified and discussed the opportunities for students to learn and reinforce Agile/DevOps concepts, e.g., ways to write better stories, deferring requirement details until needed, iterative and incremental development concepts, task estimation, progress tracking, frequent code integration, etc. For example, when there are bugs and errors after code generation, the students have to fix them by revising the story to incorporate the error messages, adjust the task estimation, and update the actual time spent fixing the bugs. Such learning opportunities might not be available when students do not attempt to create working prototypes by themselves.

3.1 Scenario 1: A Single Sprint with Stories Having Simple Description

The aim of this scenario is to quickly evaluate the capability of ADA-Gen to perform the generation of a simple web app which includes both front-end and back-end components. It is also for demonstrating the DevOps concept of continuous integration and app containerization with Docker. This scenario can be used by students to get a basic understanding of various components in a full-stack app, as well as relevant Agile and DevOps concepts such as frequent code integration and delivery of working software after each sprint.

Scenario 1 includes two simple user stories for a book listing application, namely Book List, and Book Detail. There are also two related operation stories for continuous integration and containerization of the generated app. We note that the stories do not contain much details, i.e., they lack further description, child task breakdown, and/or acceptance criteria which are usually captured from conversation with users (Cohn, 2004).

```
Scenario 1:

    Book List: I want to see the titles
     of all books displayed on the
    homepage, so that I can browse
    through the titles quickly.

    Book Detail: I want to view all the
     details of a book, namely title,
    author, and a brief summary when I
    click on the title of a book listed
     in the homepage.

    Continuous Integration: Create
    integration test cases to ensure
    that listing books and viewing book
     details work correctly. To
    implement continuous integration,
    create a YAML file that should run
    the testing code every time a
    commit is done.

    Containerization: Generate a
    Dockerfile to containerize the
    application.
```

For this scenario, ADA-Gen successfully generated a simple book listing application that lists all books and allows users to view the details of each book on a web page, without any bugs or errors. In the generated app, there were proper back-end and front-end components. ADA-Gen also generated several integration test cases, a Dockerfile, and a YAML pipeline file for Bitbucket. When the generated code is committed to the Bitbucket repository for the project, the continuous integration pipeline is run automatically. We were able to run the web app simply by using Docker commands.

This scenario shows that generating a simple full-stack web app for demonstration purpose is entirely possible with ADA-Gen (**RQ1**). It allows non-computing students to play with the code and understand key concepts in basic app components and continuous integration.

3.2 Scenario 2: Multiple Consecutive Sprints with Simple Story Description

In this scenario, the aim is to create a fairly complex full-stack web app, in particular an online book catalog. The app includes the usual CRUD operations, some business logic, user registration, login, authentication check, etc. There are more stories so they cannot be completed in a single sprint. These stories are organized into epics. However, similar to Scenario 1, the story descriptions are shallow, i.e., they lack details from conversations with user, task breakdown, and/or acceptance criteria (da Silva Neo et al., 2024). This is typical in student teams just getting started in Agile practices. As they do not have actual coding and implementation experience, they tend to over-simplify stories. This scenario is used to demonstrate that creating a proper full stack app could be difficult when you do not write appropriate user stories. It helps highlight a key learning point in Agile otherwise neglected if students do not actually create working software prototypes from user stories.

Below are the stories, their epics, and the sprint structure used. Note that story descriptions, which have a similar level of details to those in Scenario 1, are omitted due to space limitation.

```
Scenario 2:

    Sprint 1: Book List, Book Details (
   Epic: Book Management)

    Sprint 2: User Registration, Login
   (Epic: User Management), Adding
   Books (Epic: Book Management)

    Sprint 3: Adding/Removing Favorite
   Books, Favorites List Page (Epic:
   Book Management)

    Sprint 4: Most Popular Books (Epic:
     Book Management)

    Continuous Integration and
   Containerization are included in
   each sprint.
```

In this scenario, sprint 1 does not have issues with the code generation for the selected stories due to their simplicity. In the subsequent sprints, ADA-Gen generated new code and also updated the existing code to implement the new features requested in each sprint. We encountered several errors/bugs in the code generated by ADA-Gen in the later sprints. A common issue is the links to newly generated HTML pages were not included in the homepage of the app for convenient navigation. This is mainly because the user stories did not clearly specify such feature. This could serve as a learning opportunity for students to appreciate that details for stories should be fleshed out sufficiently, and that a single line story is not sufficient for actual implementation (**RQ2**).

We note that it was possible to fix the above issue during the sprint by creating a new child task or just by simply adding more description on the need to include a navigation bar with all the required links, then generating the code again with ADA-Gen. Sprint

2 had a bug involving the user registration feature, and Sprint 4 had some failed test cases due to a missing import for `generate_password_hash`. Similarly, these issues were fixed by adding child tasks containing the error messages produced during continuous integration, and re-generating the code. Through clarifying user stories by adding more tasks or descriptions when encountering errors from the generated code, students can learn to write better stories (**RQ2**).

In addition, students would be able to experience first-hand the possible technical issues occurring in a sprint. They would also be able to carry out bug fixing which requires them to look at the code and errors, and to spend actual time during the sprint. These learning opportunities could contribute to their understanding of task estimation and realistic progress tracking techniques used in Agile practices (**RQ2**). Such opportunities might not be available if they do not implement a working software in each sprint, or use a no-code platform to create apps.

Providing shallow descriptions to stories could lead to difficult issues when we need to actually realize a working software (da Silva Neo et al., 2024). Sprints 2, 3 and 4 all have the same authentication check issue, in which users could perform any CRUD operations without proper authentication. We were not able to resolve this issue using ADA-Gen given the current story structure and level of details in Scenario 2. Possible resolutions would require a complete rewriting of most of the stories to specify the child tasks such as what the front-end should return when user logins, what the back-end should check for, and so on. This scenario demonstrates that it might not be possible to create a fully working software when you do not write the proper stories as requirements. This is a good learning point for students trying to practice Agile development (**RQ2**).

In the next scenario, we would show that by writing clear stories with detailed task breakdown would enable ADA-Gen to generate better code, which we could fix to produce a fully working software at the end of each sprint.

3.3 Scenario 3: Multiple Sprints, Stories with More Detailed Description

The sprint structure and story organization here are similar to those of Scenario 2. However, more details are provided for stories, which include child task breakdown, confirmations, i.e., acceptance criteria, and the separation of front-end and back-end tasks in a story. An example is provided below.

```
User Registration: As a visitor, I want
     to register a user account using
    my email and password, so that I
    can login later.

    Frontend tasks: 1) Provide a
    registration form with fields for
    email, password, and password
    confirmation. 2) Display
    registration problems if any from
    the backend output.

    Backend tasks: 1) Implement role-
    based access control so that users
    can create accounts to login into
    the system. 2) Create a REST API
    endpoint which accepts email  and
    password to create a user account.
    3) The email has to be in a valid
    format. 4) The email must not exist
     in the database before user
    account creation. 5) The password
    has to be at least 8 characters
    which include at least one digit
    and at least one special character.
     6) The password has to be stored
    securely with hashing. 7)
    Appropriate error messages and
    status code to be returned when
    there are problems with user
    registration.
```

In this scenario, Sprint 1 and 3 had no issues with code generation. Sprints 2 and 4 had several errors that we were able to fix in a similar way used in Scenario 2. For example, the `User Registration` story required additional details to explain how to implement role-based access control by specifying the possible user roles such as `regular` and `admin`. The original user story, as shown above, did not have such details, which led to authentication implementation issues.

Halfway through the project, we were compelled to introduce more new stories to response to issues in ADA-Gen such as incomplete implementation and accidental removal of features implemented in previous sprints. These stories were not part of the original backlog. This highlights a key agile learning point in which we need to handle changes that happens during a project (**RQ2**). For example, new user stories titled `Page Accessibility` and `Personalized Content` were introduced to provide instructions on what data should be maintained after users successfully login, and how it should make use of those data to perform access control. Due to the additional stories, one more sprint was introduced in this scenario instead of just four planned initially. This demonstrates the uncertainty inherent in software projects and the application of Agile practices to build working prototype incrementally and iteratively to discover new requirements (**RQ2**). With actual development work assisted by ADA-Gen, students would be able to better appreciate this kind of challenges, and how Agile practices are used to handle them.

We noted that in Scenario 3 with better story writing, ADA-Gen was able to generate code for more complicated features like authentication and access control. However, if too much details are provided in the stories, the LLM might skip some complex instructions (He et al., 2024). This can lead to missing features during code generation. For example, in our experiments, if authentication and access control were implemented in a single story, there would be a significant amount of details regarding user registration, user login and authentication, user session management, role-based access control to restrict accesses, etc. With all these details, the LLM often failed to fully implement the correct features, or even accidentally removed previously implemented features. In our experiments with ADA-Gen, it was usually better to perform code generation and observed the bugs/errors with simple story structure first, before trying to add more details to fix these issues. This learning opportunity enables students to appreciate well-known Agile practices such as breaking down large stories into smaller ones, and deferring details until they are needed (**RQ2**).

3.4 Evaluation Summary

Summary-RQ1. From the evaluation scenarios, we have observed that it is entirely possible for ADA-Gen to generate functional full-stack web applications following a typical sprint-based software development process. The generated apps consist of proper front-end and back-end components which can be used for demonstration purposes at the end of each sprint. In this way, ADA-Gen could be useful to students without programming background in their journey to learn Agile/DevOps concepts.
Summary-RQ2. ADA-Gen provides plenty of opportunities for students to appreciate key Agile/DevOps software development practices such as writing better stories with well-defined child tasks so that working code can be generated, deferring requirement details until necessary, code review, bug fixing, continuous integration, etc. These learning opportunities would have been limited if students were not able to create working software prototypes by themselves.

3.5 Threats to Validity

We note that there are a few limitations which may affect the validity of this study. First, the output from LLMs such as GPT-4o could be non-deterministic, which may impact the consistency of the generated code. We have tried to consider this by generating and evaluating the generated apps using a number of different practical scenarios and user stories. Second, while state-of-the-art LLMs such as GPT-4o are powerful, they are unlikely to generate perfect code for all situations and requirements. Therefore it is possible that in some cases students will not be able to obtain a fully functional app for demonstration. Finally, although we role-played as a student team during the evaluation and did not attempt to modify the code manually using our programming knowledge to make the evaluation realistic, it might be possible that student teams with zero coding knowledge will find it difficult to use ADA-Gen. We plan to address this issue with a larger scale evaluation in the upcoming semester.

4 RELATED WORK

As far as we know, there is not much recent work that attempted to address the challenge of teaching Agile/DevOps to novice or non-computing students. One approach is to leverage low-code/no-code development platforms (Guthardt et al., 2024), which are visual tools enabling people without programming background to build their own software systems. The application of low-code/no-code platforms, namely Microsoft Power Apps and OutSystems, to teach Agile processes have been explored (Lebens and Finnegan, 2021; Metrôlho et al., 2020). However, we note that such platforms do not expose students to actual coding and integration issues. This may affect the learning experience and appreciation of Agile/DevOps practices in real-world software development projects.

There has been a recent trend to incorporate AI advancements into the computer science curriculum (Ozkaya, 2023) to cover real-world complex software tasks instead of just basic app development. LLMs have also been used to help students learn programming (Ta et al., 2023; Cambaz and Zhang, 2024) and write user stories (Brockenbrough and Salinas, 2024). To our knowledge, there has been no existing work making use of LLM code generation to offer a realistic teaching and learning platform for Agile/DevOps concepts.

5 CONCLUSION

In this work, we have developed a tool called ADA-Gen that performs automatic full-stack app generation in an incremental and iterative manner. ADA-Gen is designed for individuals without coding knowledge to learn Agile/DevOps software development practices. Student teams using ADA-Gen can actually execute a full software development lifecycle incrementally and iteratively. Through the usage of ADA-Gen, non-computing students can learn to appreciate important Agile/DevOps practices such as continuous integration, deferring requirement details until necessary, or to write better stories with sufficient breakdown of technical tasks so that working code can be generated via the state-of-the-art LLMs. We plan to conduct a larger scale evaluation of ADA-Gen with multiple student teams from non-computing backgrounds during the next semester at our institution.

ACKNOWLEDGMENT

We would like to thank Chen Kun and the CS480 undergraduate team SMU Zealand All Blacks for their initial contributions to this work.

REFERENCES

Al-Baik, O., Abu Alhija, M., Abdeljaber, H., and Ovais Ahmad, M. (2024). Organizational debt—roadblock to agility in software engineering: Exploring an emerging concept and future research for software excellence. *PloS one*, 19(11):e0308183.

Almeida, F. (2012). Using agile practice for student software projects. *Journal of Education and Vocational Research*, 3(9):280–290.

Amaro, R., Pereira, R., and da Silva, M. M. (2022). Capabilities and practices in devops: a multivocal literature review. *IEEE Transactions on Software Engineering*, 49(2):883–901.

Brockenbrough, A. and Salinas, D. (2024). Using generative ai to create user stories in the software engineering classroom. In *2024 36th International Conference on Software Engineering Education and Training (CSEE&T)*, pages 1–5. IEEE.

Cambaz, D. and Zhang, X. (2024). Use of ai-driven code generation models in teaching and learning programming: a systematic literature review. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, pages 172–178.

Cohn, M. (2004). *User stories applied: For agile software development*. Addison-Wesley Professional.

da Silva Neo, G., Moura, J. A. B., Almeida, H. O., da Silva Neo, A. V. B., and Júnior, O. d. G. F. (2024). User story tutor (ust) to support agile software developers. In *CSEDU (2)*, pages 51–62.

Devedžić, V. et al. (2010). Teaching agile software development: A case study. *IEEE transactions on Education*, 54(2):273–278.

Guthardt, T., Kosiol, J., and Hohlfeld, O. (2024). Low-code vs. the developer: An empirical study on the developer experience and efficiency of a no-code platform. In *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, pages 856–865.

He, Q., Zeng, J., Huang, W., Chen, L., Xiao, J., He, Q., Zhou, X., Liang, J., and Xiao, Y. (2024). Can large language models understand real-world complex instructions? In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 18188–18196.

Lebens, M. and Finnegan, R. (2021). Using a low code development environment to teach the agile methodology. In *International Conference on Agile Software Development*, pages 191–199. Springer.

Li, S., Cheng, Y., Chen, J., Xuan, J., He, S., and Shang, W. (2024). Assessing the performance of ai-generated code: A case study on github copilot. In *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*, pages 216–227. IEEE.

Mehta, K. and Sood, V. M. (2023). Agile software development in the digital world–trends and challenges. *Agile Software Development: Trends, Challenges and Applications*, pages 1–22.

Metrôlho, J. C., Ribeiro, F. R., and Passão, P. (2020). Teaching agile software engineering practices using scrum and a low-code development platform–a case study. In *Proceedings of the 15th Conference on Software Engineering Advances*, pages 160–165.

Omidvarkarjan, D., Hofelich, M., Conrad, J., Klahn, C., and Meboldt, M. (2023). Teaching agile hardware development with an open-source engineering simulator: An evaluation with industry participants. *Computer Applications in Engineering Education*, 31(4):946–962.

Oppenlaender, J., Linder, R., and Silvennoinen, J. (2024). Prompting ai art: An investigation into the creative skill of prompt engineering. *International Journal of Human–Computer Interaction*, pages 1–23.

Ozkaya, I. (2023). Application of large language models to software engineering tasks: Opportunities, risks, and implications. *IEEE Software*, 40(3):4–8.

Palopak, Y. and Huang, S.-J. (2024). Perceived impact of agile principles: Insights from a survey-based study on agile software development project success. *Information and Software Technology*, 176:107552.

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

Ta, N. B. D., Nguyen, H. G. P., and Gottipati, S. (2023). Exgen: Ready-to-use exercise generation in introductory programming courses. In *International Conference on Computers in Education*.

Wang, X., Li, B., Song, Y., Xu, F. F., Tang, X., Zhuge, M., Pan, J., Song, Y., Li, B., Singh, J., et al. (2024). Opendevin: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741*.