2023 IEEE Colombian Caribbean Conference (C3) | 979-8-3503-4179-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/C358072.2023.10436306

Empirical evaluation of automated code generation for mobile applications by AI tools

Santiago Aillon, Alejandro Garcia, Nicolas Velandia, Daniel Zarate and Pedro Wightman
School of Engineering, Science and Technology
Universidad del Rosario
Bogota, Colombia
(santiago.aillon, alejandroen.garcia, nicolas.velandia, danielan.zarate, pedro.wightman)@urosario.edu.co

***Abstract*—The rapid advancement of AI technologies has opened up new possibilities for automating various aspects of software development. Mobile app development, in particular, can benefit from AI-powered tools that assist developers in writing code more efficiently, providing suggestions, and reducing the time required for implementation. This document aims to explore how well a modern artificial intelligence tool can assist a mobile application development process. For this work, ChatGPT 3.5 was used to generate a mobile application from scratch using the Flutter framework, while the complete process was evaluated at each step. The evaluation criteria for the experience considered four indicators: code quality, solution quality, response time, and comparison with human-generated code. Results show that, up to a certain level of complexity and by using an interactive process of increasingly detailed prompts, the AI tool is capable of generating functional code, that can be the base for the inclusion of a more complex logic or structure.**

***Index Terms*—Artificial intelligence, Flutter, Mobile development, Development tools.**

I. Introduction

IN recent years, there has been a boom in the interest and development of artificial intelligence tools. Even though this technology started in the 1960s, just in the last decades it has been reaching a new golden era, where finally the computational power and the amount of data available for training, can be used to create very large models.

The accelerated advancement of language models, as well as the increase in popularity of chat systems like ChatGPT, have reached the point where these techniques are able to perform jobs that were previously considered futuristic for a machine. Many sectors of the technological industry have asked themselves what lies ahead for the future of companies and workers in these areas, given that a large number of layoffs have been observed due to the fact that these technologies, in theory, are capable of executing, in a better and more efficient way, tasks that previously required a complete team.

However, there is still much to understand about the definite impact of chat systems in the software development industry, especially given that these general models are still in development.

With this in mind, this document seeks to explore how the different artificial intelligence tools that have been developed in recent years (such as ChatGPT, CoPilot, etc.) can help in the process of developing a mobile application. For this work, the Flutter and ChatGPT frameworks will be used to evaluate and compare code written by a human and code generated by a machine, accounting for three indicators: code quality, solution quality, and response time.

II. Related work

In the last decade, artificial intelligence (AI) has revolutionized software engineering by allowing the creation of more intelligent and efficient systems. The applications of this technology have been very diverse over the years, mainly in areas of data analysis and process automation in industry, etc. However, the task of assisting humans in certain jobs, where the AI can create solutions based on a list of requirements, was still far-fetched until the arrival and popularization of Generative AI. This idea is not new. For example, in 2017, the authors of [1] highlighted the need for language models to become larger and how they should be able to support the automatic execution of everyday tasks, in order to help humans improve their productivity and reduce errors. They proposed a technique based on turning the language model into byte code, understandable directly by computers, to optimize computing resources when processing requests.

One industry that is already starting to adopt this idea is software development, reducing code writing time is always a goal in order to dedicate more time to testing and deployment. Even if the concept of frameworks is supposed to tackle that, the complexity of today's programs requires quite some time to develop both front and back-end modules, even using frameworks and some of the elements are repetitive and standard across solutions, but creating meta-frameworks should not be the solution.

Some efforts have been done to include AI in the software development process, like working on the specification and validation of requirements [2], its use to support the development of video games [3], or its application to support data analytics solutions [4], support high-level tasks like creating test plans [5], among others [6]–[8], By taking advantage of machine learning algorithms, developers can now automatically generate more efficient and reliable code than code written by humans. This is expected not just to save time, but also to improve the quality of the produced code. Exploring the code generation issue is a very recent element in the literature. In [9], the authors explore how young students faced the challenge of learning how to program in Python with and without the assistance of an AI code generator. It showed improvements in iterative prompts design and testing skills by the students. There are currently more products than studies in the area, the reason why exploratory cases like the one presented are still valuable to understand the process's advantages and limitations.

A. *ChatGPT for software development*

Chatbot GPT-3.5, or ChatGPT, developed by OpenAI, represents an advanced form of artificial intelligence known as language model [10]. It is designed to understand and generate human-like text based on the input it receives, making it exceptionally proficient in natural language processing tasks.

ChatGPT has demonstrated remarkable potential for automated software development because, with its vast knowledge base and ability to reason, it can assist developers in code generation, debugging, and even offering solutions to programming challenges. The model can comprehend and process various programming languages, allowing it to be a versatile tool for software development tasks, thereby streamlining the development process and reducing human effort.

Here are some specific examples of how ChatGPT can be used for automated software development:

- Generating code: ChatGPT can be used to generate code from natural language descriptions. This can be helpful for tasks such as generating unit tests, generating code documentation, and generating code for new features.
- Writing test cases: ChatGPT can be used to write test cases from natural language descriptions. This can be helpful for tasks such as generating regression tests, generating edge case tests, and generating performance tests.
- Analyzing test results: ChatGPT can be used to analyze test results and identify potential bugs. This can be helpful for tasks such as finding regressions, edge cases, and performance bottlenecks.

However, while GPT-3.5 holds immense promise, it still faces challenges in generating efficient and optimized code; the generated code must be used judiciously in conjunction with human expertise to ensure the production of robust and reliable software systems.

III. Methodology

The Mobile Development course at Universidad del Rosario is an elective class targeted to senior-year students who are taking their minor in Software Development. In the 2023 Spring term, a new activity was defined for the students, in order to prepare them for the upcoming scenario of having to use automated coding tools. The objective was to use ChatGPT to develop at least a module of a mobile application in Flutter. The work presented here tried to reproduce the first project in the class, which consisted of an adaptation of the well-known board game Battleship. Other groups focused on the development of UI, or modules for their final project.

The initial proposed methodology of the activity included an iterative process of creating prompts. The process starts with providing simple instructions to ChatGPT to create the application from scratch, and then, with each iteration, the prompts become more technical and detailed than the previous one. In addition, this methodology also included providing ChatGTP with the previously generated code in order to yield better results. This process allows students to understand how the tool reacted to different ways of defining requirements.

The complete process and all the indications given to ChatGPT will be shown, with a focus on the differences between the existing code and the code generated by the artificial intelligence, which is analyzed. Finally, three indicators were selected to measure the success of the final result produced by the AI tool: Quality of the code, quality of the solution, and response time. Each indicator will be given a rating on a scale from 1 to 10: 10 being the higher evaluation value, and 1 being the lowest. Students evaluated the final product based on the fact that they had already created their own versions of the application, so they could clearly identify limitations or failures in the generated code.

IV. Experimentation

A. *Requirements*

Before starting the experimentation process with ChatGPT, it is important to note some requirements that the code must have. These requirements are taken from the already existing code and will be listed below, based on their importance:

- The application must be a 5x5 battleship two-Player game. The game should have a screen where Player 1 places two ships, and another screen where Player 2 guesses the positions of the previously placed ships.
- Each ship must have dimensions of 2x1 tiles.
- The application must use the GetX library to navigate between screens.
- The application must use a GetX controller.

B. *Code generation*

As mentioned in the methodology, the first experiment consisted in asking ChatGPT to build the application without any detail. This request takes advantage of the fact that Battleship is a popular game whose rules are available and that there is a great chance that other code implementations may have existed in the training database.

1) The naïve prompt: The first prompt provided was:

"Do the code of the game Battleship with the Flutter framework" (1)

This instruction provides an initial grasp of the capabilities of ChatGPT, as well as setting a foundation to begin constructing the next instructions.

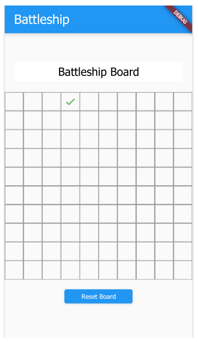

Fig. 1. Result of the first iteration of instructions

As for the macro aspects, it can be seen that ChatGPT generated an application without syntactic or run-time errors. The app consisted of a grid in which, when a tile is pressed, a pop-up dialogue will appear stating which tile was pressed. Now, regarding the micro aspects and more code-oriented elements, even though everything is in the same .dart file, each functionality has its respective method in a very organized and easy-to-understand manner.

Clearly, given that most of the requirements were not defined, the first version of the program did not consider the size of the grid, the number and size of the ships, etc. Once the starting point was clear, the next instructions will include those details to complete the requested functionalities.

2) The grid and basic requirements: The second prompt was:

"Do the code of the game battleship with a 5x5 grid where 2 ships can be placed with the Flutter framework" (2)

The result only allowed for 1x1 ships to be placed. Taking this into account, the next prompt included an emphasis on this aspect and looked like this:

"Do the code of the game battleship with a 5x5 grid where 2 ships can be placed (each ship must be 2x1 tiles) with the Flutter framework" (3)

With this modification, the dimensions of the ships were correct, but the orientation of the ships couldn't be changed, so the previous instruction was further complemented with:

"...and that allows the user to decide the orientation of the ships..." (4)

This change resulted in the inclusion of a button at the bottom of the screen which effectively changed the orientation of the ships. However, it was not intuitive at all. Therefore, it was decided to rephrase the instruction, to emphasize the desired logic of the program, which resulted in the instruction:

"Create the code for an application of the game Battleships using a 5x5 grid where two ships can be placed. Each ship must have a size of 1x2. When the user taps on a cell, the first half of a ship will be placed, and when they tap again on an adjacent cell, the other part of the ship will be placed. All of this using the Flutter framework" (5)

This last instruction yielded two main errors: A syntax error and a logic error. The syntax error occurred due to Flutter's `Null` control, but it was easily solved using `late`. This error appeared because the version of Dart and Flutter that ChatGPT was trained with is an older version that did not include null protection, thus the current interpreter threw that exception.

On the other hand, the logic error arose because it only allowed changing the orientation of the ships when they were positioned on one of the edges of the grid. Taking this into account, the previous instruction was slightly altered to get the desired result.

"Create the code for an application of the game Battleships with a 5x5 grid where two ships can be placed. Each ship must have a size of 1x2. So, no matter the coordinates the user selects, when they tap on a cell, the first half of a ship will be placed. Then, when they tap again on an adjacent cell, the other part of the ship will be placed. All of this using the Flutter framework." (6)

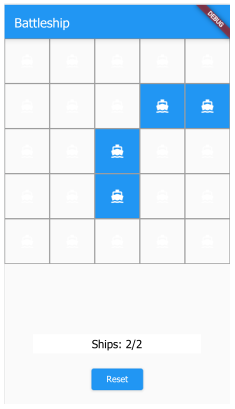

Fig. 2. Result of instruction (6)

This outcome generated a viable version of the first Player board, that allowed the positioning of the ships, after 6 prompts. The next step is to integrate the functionality for two Players.

3) Two-Player game: The Two-Player game required a more detailed explanation of the desired logic. Thus, the previous instruction was modified to add the feature of allowing

Player 1 to place the ships and Player 2 to guess the positions of the ships set by Player 1. The resulting prompt was as follows:

"Create the code for an application of the game Battleships with a 5x5 grid. The game should allow two Players; Player 1 must place 2 ships, each with 2x1 dimensions, and each ship must be placed in halves. That means, when Player 1 taps on any cell, one half of the ship should be placed, and when they tap on another adjacent cell, the entire ship should be placed. After Player 1 has positioned both ships, the application should navigate to another screen where Player 2 can guess the positions of the ships placed by Player 1. When Player 2 has guessed the positions of both ships, the application should display a screen announcing that Player 2 won, with a button to restart the entire game from scratch. However, if Player 2 fails to find both ships, the application should display a screen announcing that Player 2 lost, with a button to restart the entire game from scratch. All of this should be done using the Flutter framework." (7)

This new instruction brought about new issues: Firstly, for the buttons, ChatGPT used the `RaisedButton()` widget, which was declared deprecated in May 2021. However, this was easily solved by replacing all occurrences of `RaisedButton()` with `ElevatedButton()`. Secondly, all attributes of the classes were not handling `Null` correctly, so a `late` keyword was added to solve this issue. And finally, and most importantly, even though the application effectively allowed Player 1 to position the ships and Player 2 to guess their positions, the AI overlooked the fact that the ships must be of size 1x2.

Further tests were conducted by modifying the wording of instruction 7 and regenerating the responses from ChatGPT multiple times, but the same errors persisted. Six different re-phrasings were attempted for this instruction, and in each case, the response was regenerated three times but the AI still overlooked some of the instructions made earlier.

Given the situation, it was decided to rethink the methodology by which instructions were given to the AI. Now, instead of building upon the previous instructions, the code that was previously generated will be passed to ChatGPT in order to have the AI build upon the previously generated code. Therefore, the code generated in instruction 6, which had the Player 1 screen fully functional, was passed back to ChatGPT with the following instruction:

"Use the previous code to create an application of the game Battleships. The application should allow two Players. Player 1 must place the ships (code sent previously). Player 2 must guess the positions where Player 1 placed the ships." (8)

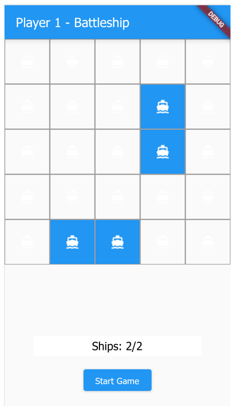

Fig. 3. Result of prompt 8 for the implementation of the two-Player functionality on the Player 1 screen

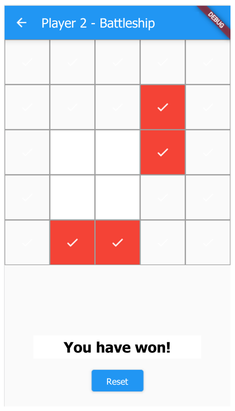

Fig. 4. Result of prompt 8 for the implementation of the two-Player functionality on the Player 2 screen

The instruction implemented the functionality of two Players without any syntax errors. However, during Player 2's turn, it only allowed pressing the cells where Player 1 had positioned the ships, meaning there was no possibility for Player 2 to lose. To address this issue, ChatGPT was given the following correction:

Modify the previous code so that during Player 2's turn, when they tap on a cell incorrectly, the cell turns orange, but when they tap on a cell containing a ship, it turns green. Player 2 has only 6 attempts to guess the positions. (9)

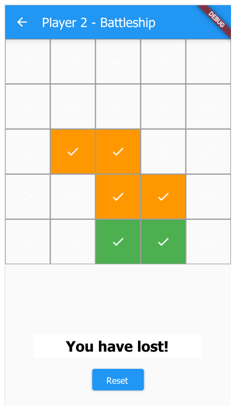

Fig. 5. Result of the implementation of Player 2's losing screen

Now, for the last correction to finish with the first requirement, it was necessary to fix the reset button. When this button is pressed at the end of Player 2's turn, it returns to the Player 1 screen, but both ships positioned by Player 1 in the previous attempt remained in their original positions. Therefore, ChatGPT was instructed as follows:

"Correct the previous code so that when Player 2 presses the reset button, it goes back to the Player 1 screen with the ship positions reset." (10)

And with this, everything necessary to fulfill the first requirement is complete. Now, we will ask ChatGPT to modify the previously generated code so that the application changes screens using the GetX library.

4) Including the GetX library: This step required the inclusion of the GetX library in two large functionalities: screen navigation and creating a controller for the game. So, the first function was pretended to be included with the next prompt:

"Change the previous code so that the application uses the Flutter library GetX only to switch between screens." (11)

This result brought back errors such as Player 2 not being able to lose or the game not resetting properly. After five instructions to solve these issues, the desired outcome was achieved. It is important to mention that every time ChatGPT was ordered to use the GetX library, it made all screen changes correctly by passing the required attributes. However, it left the `MaterialApp()` as it was, so it had to be manually changed each time the tests were performed to `GetMaterialApp()`. This way, the second requirement was fulfilled.

Finally, for the third and final requirement, the used prompt was the following:

Change the precious code and make it work properly using GetX controllers. (12)

This last instruction brought many errors, including syntax errors, run-time errors, and logic errors. Ten more instructions were given to the AI to modify and correct the errors from the code generated by instruction 12, but the errors persisted. The errors included variables without `Null` safety, methods of controllers being invoked without being declared before, and the use of the `Obx()` widget without any observable variables, among others.

After conducting tests and considering the logic errors from integrating the GetX controller, it was decided to keep the code generated just before instruction 12 as the final version, which means, only fulfilling the first two requirements. This will be the code analyzed in the next section.

V. Results

The code of the final successful application delivered by ChatGPT, although it did not compile on the first attempt, once all the errors were fixed, each requirement (except the last one) were implemented without any issues. Taking this into account, first, a comparison will be made with the existing code, focusing on aspects of logic and organization. Then, an analysis of the process to achieve the desired result will be conducted, followed by an evaluation of the three indicators mentioned in the methodology.

Performing a comparison with the existing code of the game, it can be noticed that one of the main differences is the separation of game functionalities into widgets. In the existing code, each main functionality of the Player screens is created as a separate widget and then assembled in each screen's class, which facilitates the debugging process and improves the overall code organization. However, the code provided by the AI does not follow this practice, having all the functionalities of a screen in a single widget. Although this makes the code shorter, it also makes it less understandable. At the time of experimentation, ChatGPT is unable to divide each widget, classes, and functions into separate files, which also negatively impacts the code's organization. Nevertheless, despite these organizational errors, the code itself is quite comprehensible, which was a pleasant surprise.

Even though the organization is not the strong point of the code generated by ChatGPT, many logical implementations of the ship positioning system and turn control were surprisingly clever compared to the existing code.

Now, although the obtained result is very close to the desired outcome, the process to reach this result turned out to be arduous and impractical. When instructions were given to the artificial intelligence, it was evident that it understood what was requested as ChatGPT would write a small paragraph summarizing and explaining all the functionalities of the code. However, when testing the code, it often did not work correctly. For instructions with simpler functionalities, the code provided by ChatGPT had simple `null` control errors when declaring variables. However, as the complexity of the functionalities to be implemented increased, errors such as the use of an undeclared function began to appear, which, for the purpose of this research, was considered too severe to correct.

It is important to note that these results and observations do not mean that the tool is not useful; quite the contrary. It proved to be a useful tool to generate simple code that could fulfill some of the basic requirements and that could be used as a starting point to start implementing more complex functionalities that go beyond its current capabilities. 4 simple indicators were defined to evaluate the characteristics of the experience:

- Quality of the code: 6/10. Despite making quite a few logical errors, the syntax errors were relatively easy to fix. Additionally, although the code it generates is not properly organized it is still easy to understand. The tool was able to separate the code into different files to avoid having everything in the main.dart.
- Quality of the solution: 7/10. The quality of the final application is quite good; it is functional and intuitive, which are considered of utmost importance. The application fulfills almost all the requirements, except the inclusion of GetX controllers.
- Time consumption: 6/10. The process of creating an application from scratch solely with code provided by ChatGPT can become lengthy and tedious. Although it is evident that ChatGPT understands what needs to be done, it does not always execute it perfectly, which is why many multiple iterations and adjustments were necessary to achieve the desired result.
- Comparison with the existing code: 6/10. As evidenced in the experimentation, even when ChatGPT generated code that partially worked, it only generated a single main file. This makes code maintenance more challenging. Additionally, it produced various uses of deprecated methods and did not handle null variables correctly, among other errors previously mentioned.

Based on the scores, the average performance is 6.25. This methodology was defined during the class exercise, but for a future work, not only more tools will be included but more detailed evaluation criteria.

VI. Conclusions

ChatGPT is an extremely powerful tool that can generate code snippets and assist developers in writing code more efficiently. However, developers should be aware of its limitations. The quality of the generated code may not always be reliable and may require human intervention to correct errors and ensure that the code meets the specified requirements. Thus, to be able to use tools like these, it is necessary to understand the language or framework in which ChatGPT is instructed to generate the code and have enough experience with programming logic.

The experiment showed how defining the requirements is a key element to using this tool effectively. An iterative approach seems to be the way to evolve the generated code, by including new functionalities, one at a time. This also gives the chance to the programmer to verify the code, identifying errors, logic failures, and lack of compliance with the requirements, among other issues. Human intervention and expertise are still essential for ensuring the quality and accuracy of the generated code.

In addition, given that ChatGPT 3.5 was used for this experiment, which does not have internet access, a significant difference was evident when working with Flutter libraries and working without them. When not requesting the artificial intelligence to work with any library, the errors were trivial or minimal. However, when working with a library that required higher-order logic, greater errors started to appear, likely due to the AI not having access to the latest library updates and documentation. It is important to mention that this aspect may be addressed with the use of ChatGPT 4.0, which could potentially have access to more up-to-date information and overcome some of these limitations. In future work, other tools can be examined, like Google's Bard, GitHub's Copilot, among others to compare their code generation capabilities.

Acknowledgment

ChatGPT v3.5 and Google's Bard were used to support writing excerpts of the Related Work section. The final generated code can be found at https://github.com/tatoGtato/Battleships-using-AI-tools .

References

[1] A. Trifan, M. Angheluş, and R. Constantinescu, "Natural language processing model compiling natural language into byte code," in *2017 International Conference on Speech Technology and Human-Computer Dialogue (SpeD)*, 2017, pp. 1–6.

[2] W. Haider, Y. Hafeez, S. Ali, M. Jawad, F. B. Ahmad, and M. N. Rafi, "Improving requirement prioritization and traceability using artificial intelligence technique for global software development," in *2019 22nd International Multitopic Conference (INMIC)*, 2019, pp. 1–8.

[3] M. O. Riedl and A. Zook, "Ai for game production," in *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, 2013, pp. 1–8.

[4] M. Marinho, D. Arruda, F. Wanderley, and A. Lins, "A systematic approach of dataset definition for a supervised machine learning using nfr framework," in *2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC)*, 2018, pp. 110–118.

[5] V. Bilgram and F. Laarmann, "Accelerating innovation with generative ai: Ai-augmented digital prototyping and innovation methods," *IEEE Engineering Management Review*, vol. 51, no. 2, pp. 18–25, 2023.

[6] S. B. Pandi, S. A. Binta, and S. Kaushal, "Artificial intelligence for technical debt management in software development," 2023.

[7] M. Mrak, M. R. Hashemi, S. Shirmohammadi, Y. Chen, and M. Gabbouj, "Editorial applied artificial intelligence and machine learning for video coding and streaming," *IEEE Open Journal of Signal Processing*, vol. 2, pp. 410–412, 2021.

[8] N. D. Q. Bui, H. Le, Y. Wang, J. Li, A. D. Gotmare, and S. C. H. Hoi, "Codetf: One-stop transformer library for state-of-the-art code llm," 2023.

[9] M. Kazemitabaar, X. Hou, A. Henley, B. J. Ericson, D. Weintrop, and T. Grossman, "How novices use llm-based code generators to solve cs1 coding tasks in a self-paced learning environment," 2023.

[10] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," *CoRR*, vol. abs/2005.14165, 2020. [Online]. Available: https://arxiv.org/abs/2005.14165