

# Visualizing Source Code as Comics Using Generative AI

David Heidrich

*Institute for Software Technology  
German Aerospace Center (DLR)  
Weßling, Germany  
david.heidrich@dlr.de*

Andreas Schreiber

*Institute for Software Technology  
German Aerospace Center (DLR)  
Köln, Germany  
andreas.schreiber@dlr.de*

**Abstract**—The architecture and inner structure of software is often only implicitly available in the form of its source code and thus not tangible and intuitively easy to understand for non-programmers and laymen. Our goal is to create visualizations as automatically as possible, with which such people can nevertheless understand the software or parts of the software and get a feel for the structure of the software and how its methods work. Especially for newcomers to software projects, for management or even for students and pupils, it can be helpful to get a non-technical insight into the software.

We use the concept of visualizing information as comics to present aspects of the software as strikingly as possible, as comics are an effective way to present complex systems and interrelationships for certain target groups. For this purpose, we present a method to generate comics from source code. Our semi-automated process is based on generating a prompt for an LLM from source code, which in turn generates a prompt for a comic image generation using the text-to-image model Stable Diffusion. We show that generative AI methods can be used to rapidly generate human-compatible artistic representations from source code. However, further research is needed to validate the understandability of the results.

**Index Terms**—visualization, software visualization, comics, generative ai, stable diffusion

## I. INTRODUCTION

Visualizing complex systems using comics is an effective way to communicate difficult scenarios and messages to an audience (e.g., in science communication [1]). Some techniques for visualizing complex systems with comics are:

- Understand perception and cognition: A solid understanding of perception and cognition is vital to creating good visualizations.
- Simplify details: Sometimes details can get in the way of understanding complex systems. It is important to simplify and focus on the most important aspects.
- Use comic strips: Comic strips can be used to represent certain activities in a complex system, such as the provenance of personal health data [2] or to visualize and communicate changes in dynamic networks [3].
- Highlight trends and patterns: Visualization is a powerful tool that can clarify complex messages and highlight trends or patterns in lots of data.
- Use modular information design: A modular approach to information design can help anyone visualize complex issues in an attractive and innovative way.

Overall, the goal of visualizing a complex system is to make sense of people’s experiences in the context of the whole [4]. A benefit of using comics to visualize complex systems is that by combining visualization with metaphors and character-driven narratives, comics have the potential to make complex information more accessible and engaging [1].

In our work, we aim to apply the concept of visualizing information as comics to software visualization to present aspects of software in a striking way. We present a method for generating comics from source code. Our semi-automatic process is based on generating a prompt for a pre-trained *Large Language Model* (LLM) from the source code, which in turn generates a prompt for a comic image generation using a *text-to-image stable diffusion model*. The background information and our contributions are structured as follows:

- We give a brief overview of software visualization and methods of generative AI as far as it is relevant for our work. (Section II).
- We describe the method, illustrate it with an example and discuss the preliminary result (Section III).

## II. BACKGROUND

### A. Software Visualization

Software visualization [5] can help humans (e.g. developers) to understand complex software systems or to get insights into their behavior and quality properties. For example, visualizations are often used to understand source code [6], such as through various types of diagrams that depict the source code and its structure in an easy-to-understand way. Other use cases for software visualization includes areas such as becoming familiar with an unfamiliar software system [7] or performing reverse engineering and, maintenance and debugging tasks [8]–[10].

### B. Generative AI

Generative artificial intelligence (AI) enables machines to create content similar to that created by humans. It is capable of creating novel, original, and coherent outputs based on patterns it has learned from its input data. This growing set of tools augments human creativity by, for example, creating realistic images [11], creating realistic videos [12], writing persuasive text [13], or composing music [14].

An increasing number of open-source LLMs focus on software engineering tasks, such as code generation (e.g., *StarCoder*) or comprehension (e.g., *StarChat Alpha*).

### III. GENERATING COMICS FROM SOURCE CODE

Our semi-automated method of converting source code to comics uses two different generative AI methods. We use the transformer-based language model *GPT-4* [15] for text generation and the latent text-to-image diffusion model *Stable Diffusion* [11] for image generation<sup>1</sup>.

Our method consists of four steps (see Figure 1):

- (1) **Preparation** The designer creates instructions for the LLM.
- (2) **Text Generation** The LLM generates prompts from the instructions.
- (3) **Image Generation** The stable diffusion checkpoint generates the individual frames (i.e., comic per region).
- (4) **Finalization** The result is finalized, for example, up-scaling, denoising, and detailing (e.g., speech bubbles).

In the preparation step, the designer chooses the detail level of the resulting visualization and prepares the text prompts for the text generation. For example in a C# software project, each comic frame could represent a class, which gives an overview of the namespace, or each comic frame could represent a method, which gives an overview of the class. Note that—especially for bigger software systems—we must also consider the available context length of the used LLM, which determines the maximum length of an instruction (i.e., the amount of source code that we can process). In the following we will focus on representing methods with comic frames. After selecting the detail level, the designer prepares the text prompts. Note that the text prompt starts with the command *Explain to a five year old*, which is commonly used to simplify the output [16]. To get a description of the character, which should be based on properties of the selected class, we use:

```
Explain the functionality of the following C#
class to a five year old. Explain the class as
it is a cartoon character. Map important pro-
perties of the source code to visuals of the
cartoon character. Give a detailed description
of the cartoon character.
[INSERT CLASS SOURCE CODE]
```

To get a description for each frame, which should be based on the properties of the child components, we use the following statement:

```
Convert every method of the class to a comic
frame. Every frame should explain to the five
year old what is happening in the method.
Describe every frame in detail. The character
you just described should be in every frame.
```

<sup>1</sup>In practice, we use the *Stable Diffusion web UI* <https://github.com/AUTOMATIC1111/stable-diffusion-webui> with the style checkpoint *Counterfeit-V3.0* <https://civitai.com/models/4468/counterfeit-v30> (based on the *Stable Diffusion 1.5* base-model <https://huggingface.co/runwayml/stable-diffusion-v1-5>)

The training data of current LLMs, such as *GPT-4*, does not contain knowledge about text prompts for image generation. Hence, until future LLMs can directly generate the prompts for image generation, we extract stable diffusion prompts from *GPT-4*'s output by hand. Here, the artist can also add more details and some default stable diffusion prompts, like *masterpiece* or *beautiful*. We then use the text prompts to generate the individual comic frames inside the *Stable Diffusion web UI*. We use the *img2img Inpaint* feature to first draw a mask on the area where the comic frame should be created and then generate the specific comic frame. Here the artist can generate multiple outputs using different seeds and select the image that they like the best. They repeat the process for each comic frame until the basic comic is complete. To finalize the comic, the artists can upscale the comic and add more details, like method names or speech bubbles.

#### A. An Illustrative Example—A Comic Picture for a C# Constructor Class

As an example, we applied the method to visualize a constructor class of the open-source C#-based software system *IslandViz VR* [17], [18]: The constructor *DockGOConstructor* has 214 lines of code and contains 5 methods. It is responsible for creating and placing dock *GameObjects* into the visualization. The first instruction given to *GPT-4* consisted of 4,220 tokens. While *GPT-4* can handle this request with a maximum context length of 8,192 tokens [19], this request would already exceed the limit of, for example, current *LLAMA* models with a maximum context length of 2048 tokens [13]. Our first instruction resulted in the following *GPT-4* output:

```
This class, called DockGOConstructor, is like
a super builder cartoon character. Let's call
him "Builder Bob". Bob is not just any builder
though; he's a magical builder who can create
and place "docks" around little islands.
These docks represent connections to other is-
lands, showing who needs help from whom.
```

```
Builder Bob has a few tools and features that
help him do his job.
```

```
1. **Tool belt (private variables)**:
Builder Bob has a tool belt with his important
tools. He has a 'status' tool that tells him
if he's working, resting, or done with his
work. Then he has a 'cb' tool, which is like a
special phone to call his boss when he's
finished his work. [...]
```

```
2. **Special abilities (methods)**:
Builder Bob has some cool abilities or actions
he can perform [...]
```

```
So, if you think about it, Builder Bob is real-
ly cool. He not only builds things but also
organizes them perfectly so everyone can reach
who they need to. He's the magical builder
hero of our cartoon adventure!
```

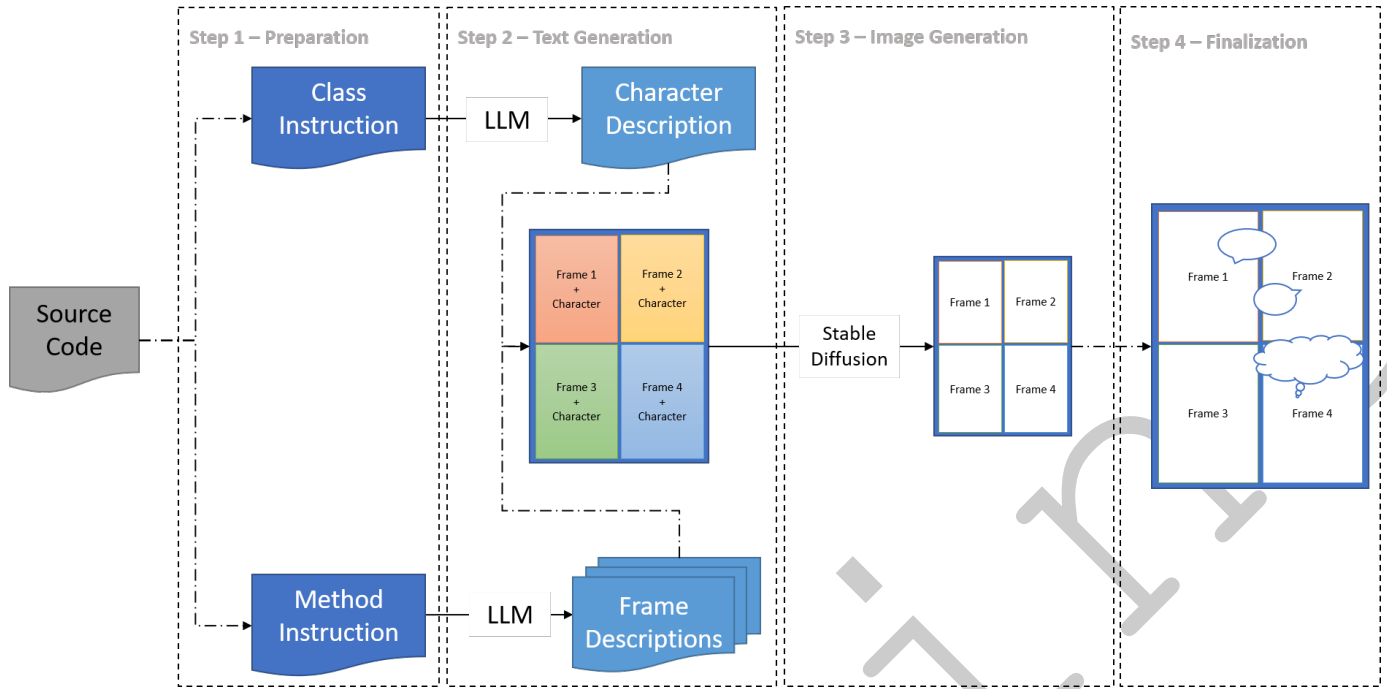


Fig. 1. The generation process from source code to comics steps (1)–(4). Dotted arrows indicate non-automated steps.

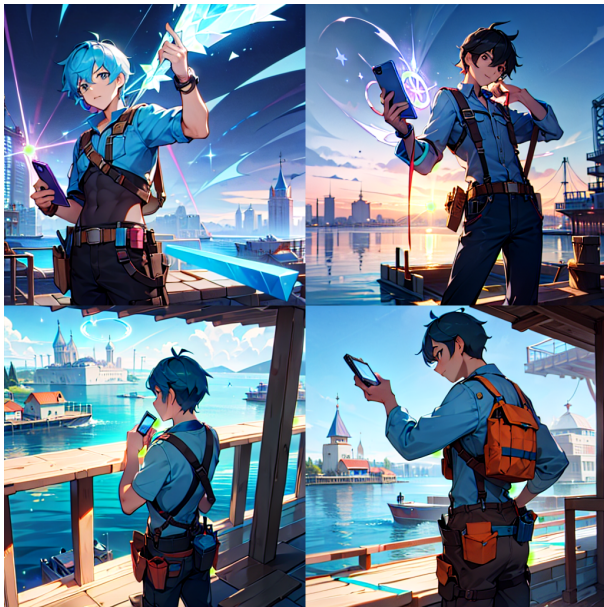


Fig. 2. Output for character description using four different seeds.

From this description, we then extracted the character prompts: *best quality, 1man, (((magical builder))) with tool belt and phone, islands and docks in background* and generated results using this prompt (Fig. 2). Our next step was to give GPT-4 the second instruction. This resulted in the following output:

1: Start Method  
The first panel shows Builder Bob just waking

up, stretching, and yawning. He's ready to start the day. He looks at his status watch, which reads "Idle", signifying he's ready to go. He checks his big bag labeled "VisualizationContainer" to ensure it's empty and ready to fill. He also takes out his dock-List notepad, flips through it to ensure it's blank, ready for a new day of work.

Panel 2: Construct Method  
In this panel, we see a line of cartoon islands eagerly waiting for Builder Bob. Bob receives a list of islands and a special phone (the 'cb' tool). He sets his status watch to "Working" and announces to everyone, "Started with Dock-GameObject construction!". He waves his hand, and we see a sparkling magic swirl around it, symbolizing the start of his magical coroutine to construct docks for all islands.

Panel 3: ConstructAll Method - Part 1  
Builder Bob is now seen hard at work. There's a magical portal in front of him, from which one island at a time appears. Bob is building docks one after another. Each time he finishes a dock, the island with its new dock disappears back into the portal, and a new one appears.

Panel 4: ConstructAll Method - Part 2  
Now, Bob is linking the docks together. He's depicted drawing magical arrows from one island's dock to another's, illustrating the connections between them. Once he's done, he makes a triumphant "Finished with Dock-Game-Object construction!" announcement, sets his status watch to "Finished", and uses his 'cb' phone to inform his boss that he's done.

Panel 5: FindSuitablePosition2D Method  
 This panel features Bob with a magical compass and a map, surrounded by islands and docks. There's a humorous sequence of him using the compass to find the perfect spot for a new dock. Each time he thinks he has found a spot, a comic-style "BONK!" appears as the compass refuses the position, until finally, it gives a big "DING!", indicating the perfect spot.

Panel 6: ConstructDockGO Method  
 The final panel is a culmination of all the previous panels. Builder Bob, using his magical powers, creates a dock, adjusts its size, and places it in the perfect spot (as identified in the previous panel). This panel is filled with magic, excitement, and a sense of achievement, as Bob successfully completes his task. [...]

We then extracted the comic frame prompts from this output (Table I) and generated the individual comic frames. To do this, we started with an empty white image and used the `img2img Inpaint` function to mask out the area where the frame should be and then generated a batch of potential frames (Fig. 3). We then selected one of the generated images and repeated the process from there. Finally, we scaled up the images resolution and added details in form of method names and speech bubbles, based on GPT-4's description. At the end, the final comic (Fig. 4) shows the individual activities that the class can perform, with the class itself corresponding to the main character of the comic.

TABLE I  
 PROMPTS USED TO GENERATE THE EXAMPLE COMIC FRAMES.

Frame	Image Generation Prompt
1	<i>(best-quality:0.8), 1man, (((magical builder))) with tool belt, waking up, sunrise, looking at notepad, big bag, docks and little islands in background</i>
2	<i>(best-quality:0.8), 1man, (((magical builder))) with tool belt waving his hand, (islands in background), phone, sparkling magic swirl</i>
3	<i>(best-quality:0.8), 1man, (((magical builder))) with tool belt, islands appearing from (magical portal)</i>
4	<i>(best-quality:0.8), 1man, (((magical builder))) with tool belt linking docks together, islands in background, arrows</i>
5	<i>(best-quality:0.8), 1man, (((magical builder))) with tool belt, magical compass, map, surrounded by islands</i>
6	<i>(best-quality:0.8), 1man, (((magical builder))) with tool belt placing a dock in the perfect spot, magic, proud, happy</i>

## B. Discussion of Preliminary Results

Our method is capable of semi-automatically generating comics from source code. While we have automated most of the comic creation process, it still keeps the creator in the loop. In stable diffusion, the use of different seeds produces a wide range of overall similar images. This preserves the creative freedom of the designer who can choose from multiple outputs



Fig. 3. Output for the first frame using four different seeds.

while also making sure that important details are present in the output. However, repeatedly asking the same question in GPT-4, i.e., using different seed, results in distinctly different GPT-4 outputs. In our example, we always used the first output. But asking for the character again resulted in an octopus character for the second request and a duck pirate character in the third request. This indicates the need for more detailed instructions in order to achieve more coherent GPT-4 outputs.

To allow creators without prompting experience to generate the stable diffusion prompts, we could automate more steps, such as extracting stable diffusion prompts from GPT-4 output by using an instruction that explains the concept of stable diffusion prompting. This, however, requires a long enough content length. Additionally, we could reduce the number of user inputs further by inserting all prompts at once by using *regional prompting* [20].

To validate the methodology and our results, we could conduct user studies with software developers rating comics created by artists with no technical experience. Alternatively, since our methodology allows software developers to create such comics themselves, we could also compare design choices (e.g., which frame did they think fitted best) between the two user groups.

## IV. CONCLUSION AND FUTURE WORK

Combining multiple generative AIs holds great promise for improving software visualization. As LLMs are trained on more recent data and image generation models become capable

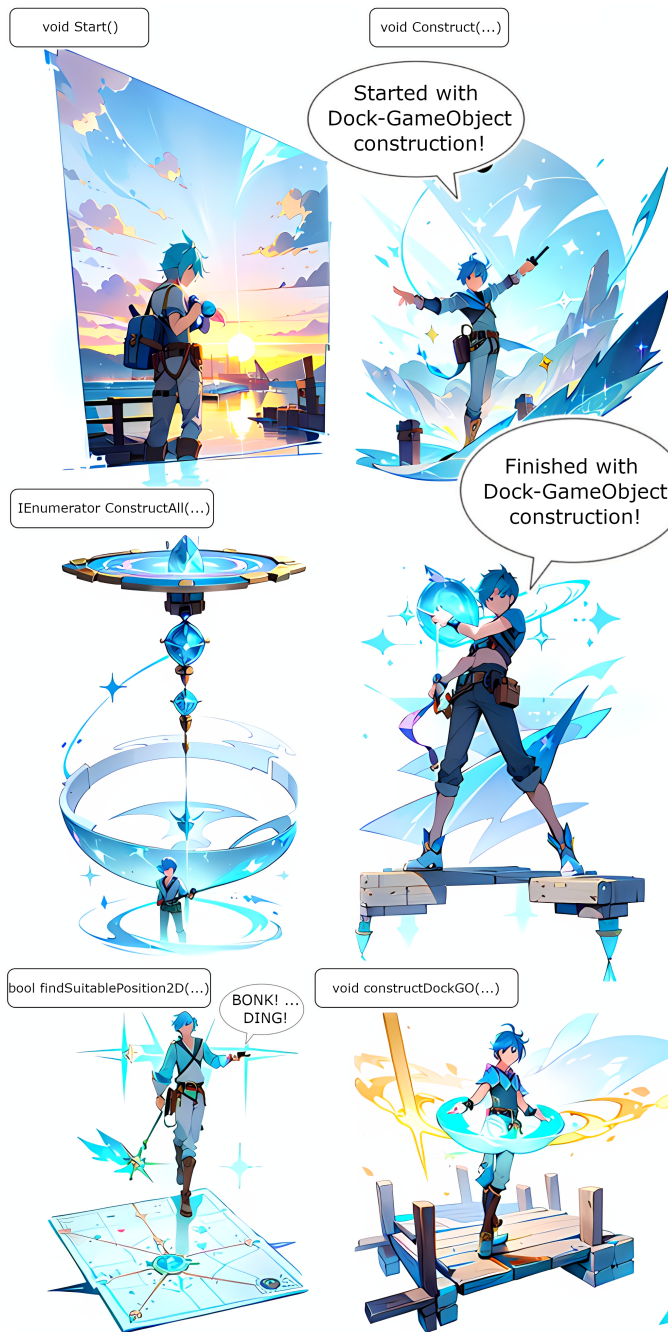


Fig. 4. The final comic shows strikingly the individual activities that the class can perform. The class itself corresponds to the main character. You can see how the class starts a new day, how it starts creating new objects, how it makes islands appear in front of it, how it chooses a position on a world map, and how it stands on the final dock casting spells.

of displaying coherent text within their generated images, we plan to fully automate the generation process. We also plan to replace GPT-4 with an open source language model.

We also want to explore the use of generative AI to automatically generate code flow diagrams or data flow diagrams based on source code. This could reduce manual effort and ensure a more accurate, up-to-date representation of the software

system. Such dynamic visualizations could even adapt in real time, which would be invaluable for understanding and debugging complex software behavior as it is being built. Additionally, generative AI could also generate content for visualization systems or learning and support environments based on gamification or serious games [21].

## REFERENCES

- [1] M. Farinella, "The potential of comics in science communication," *Journal of Science Communication (JCOM)*, vol. 17, no. 1, 2018.
- [2] A. Schreiber and R. Struminski, "Visualizing the provenance of personal data using comics," *Computers*, vol. 7, no. 1, 2018.
- [3] B. Bach, N. Kerracher, K. W. Hall, S. Carpendale, J. Kennedy, and N. Henry Riche, "Telling stories about dynamic networks with graph comics," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, ser. CHI '16. New York, NY, USA: ACM, 2016, pp. 3670–3682.
- [4] D. Bihanic and T. Polacek, "Models for visualisation of complex information systems," in *2012 16th International Conference on Information Visualisation*, 2012, pp. 130–135.
- [5] L. Bedu, O. Tinh, and F. Petrillo, "A tertiary systematic literature review on software visualization," in *2019 Working Conference on Software Visualization (VISSOFT)*, 2019, pp. 33–44.
- [6] B. A. Price, R. M. Baecker, and I. S. Small, "A principled taxonomy of software visualization," *Journal of Visual Languages & Computing*, vol. 4, no. 3, pp. 211–266, 1993.
- [7] L. von Kurnatowski, D. Heidrich, N. Güden, A. Schreiber, H. Polzin, and C. Stangl, "Analysing and visualizing large aerospace software systems," in *ASCEND 2021*, 2021, p. 4082.
- [8] E. R. Gansner and S. C. North, "An open graph visualization system and its applications to software engineering," *Software: practice and experience*, vol. 30, no. 11, pp. 1203–1233, 2000.
- [9] R. Koschke, "Software visualization in software maintenance, reverse engineering, and re-engineering: a research survey," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 15, no. 2, pp. 87–109, 2003.
- [10] D. Holten, "Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data," *IEEE Transactions on visualization and computer graphics*, vol. 12, no. 5, pp. 741–748, 2006.
- [11] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022, pp. 10684–10695.
- [12] U. Singer, A. Polyak, T. Hayes, X. Yin, J. An, S. Zhang, Q. Hu, H. Yang, O. Ashual, O. Gafni *et al.*, "Make-a-video: Text-to-video generation without text-video data," *arXiv preprint arXiv:2209.14792*, 2022.
- [13] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "Llama: Open and efficient foundation language models," 2023.
- [14] A. Agostinelli, T. I. Denk, Z. Borsos, J. Engel, M. Verzetti, A. Caillon, Q. Huang, A. Jansen, A. Roberts, M. Tagliasacchi *et al.*, "Musiclm: Generating music from text," *arXiv preprint arXiv:2301.11325*, 2023.
- [15] OpenAI. (2023) GPT-4. [Online]. Available: <https://openai.com/>
- [16] K. Jeblick, B. Schachtner, J. Dextl, A. Mittermeier, A. T. Stüber, J. Topalis, T. Weber, P. Wesp, B. Sabel, J. Ricke *et al.*, "Chatgpt makes medicine easy to swallow: An exploratory case study on simplified radiology reports," *arXiv preprint arXiv:2212.14882*, 2022.
- [17] M. Misiak, A. Schreiber, A. Fuhrmann, S. Zur, D. Seider, and L. Nafeie, "Islandviz: A tool for visualizing modular software systems in virtual reality," in *2018 IEEE Working Conference on Software Visualization (VISSOFT)*. IEEE, 2018, pp. 112–116.
- [18] M. Misiak, Rawi85, and A. Schreiber, "Dlr-sc/island-viz: Islandviz 1.0," Oct. 2018. [Online]. Available: <https://doi.org/10.5281/zenodo.1464633>
- [19] OpenAI. (2023) Models documentation. [Online]. Available: <https://platform.openai.com/docs/models/gpt-4>
- [20] hako mikan. (2023) Regional prompter. [Online]. Available: <https://github.com/hako-mikan/sd-webui-regional-prompter>
- [21] R. Gökmen, D. Heidrich, A. Schreiber, and C. Bichlmeier, "Stereotypes as design patterns for serious games to enhance software comprehension," in *2021 IEEE Conference on Games (CoG)*, 2021, pp. 1–3.