The Effectiveness of Style Guides for

Tsuboniwa: A Garden Dream
An Audio-Visual Virtual Reality Experience
A supplementary document to the “Tsuboniwa: A Garden Dream” MQP Report

A Major Qualifying Report
submitted to the faculty of the
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Bachelor of Science

By:
Elaine Chen
Madeline Perry

Project Advisors:
Jennifer deWinter
Charles Roberts
Yunus Telliel

Project Sponsor:
Atticus Sims, Kyoto VR

Date: 12/16/2021

This report represents the work of one or more WPI undergraduate students submitted to the faculty as evidence of completion of a degree requirement. WPI routinely publishes these reports on its web site without editorial or peer review.
*Tsuboniwa: A Garden Dream* is a project that created a new code base, where developers will contribute after we complete our MQP. Because this is a new codebase, documentation with explanations about how the code works is vital to the future development of the project. This report provides a literature review analyzing how current documentation practices are not sustainable and cause problems for future developers contributing to open source software. It provides a structure and reasoning as to why a style guide is the most effective documentation practice for *Tsuboniwa: A Garden Dream*. This structure is based on analysis of the most effective practices from various tech companies. This literature review and analysis of style guides provides support and reasoning to why the proposed structure would be the most beneficial for the style guide.

*Tsuboniwa: A Garden Dream* is a Virtual Reality (VR) experience in the first stages of development, aimed to be built upon as open source software. This project is utilizing C# programming, and will require further development in the future. Because we only developed the first three stages of the experience, future stages will need to be compatible with the current code and issues may arise as development continues. Additionally, the style guide should provide explanations on how the project utilizes Unity as a development platform. This supplementary appendix is written to provide support and reasoning as to why current documentation practices for open source software are not informative enough for future developers and need to be improved.
Software development is a new avenue of study born from the age of computers and technology (Cereceda, 2020). It utilizes coding languages to create programs for people to use in their daily life or for specific tasks. Using different coding languages requires a significant amount of knowledge in understanding what certain functionalities do and how to alter code that has caused errors, known as bugs (Cereceda, 2020). Because coding languages can be difficult to understand without the thought process known when creating a program, it can be difficult to comprehend written code, especially for new programmers.

One reason for the booming popularity and growing community of software development is the usage of open source software, where developers publish code publicly for others to build on or reference (Lenarduzzi, 2020). This development practice grew in popularity because developers are able to build on previous work, rather than starting from scratch every time they start a new project. This process promotes collaboration during development, which has been utilized in industry practices to create better products with more ideas being circulated (Cereceda, 2020). However, during development, there is a lack of written documentation explaining what functions mean, the reasoning behind them, or even what certain programs do. This lack of updated documentation creates a disconnect for incoming developers. Additionally, software development is constantly updated for new features or edited to fix errors. This causes code to change and documentation is often neglected and becomes outdated. While it is good practice to have updates and an improved program, it causes issues for new developers who are unfamiliar with the codebase.
Graduate students are a group of students that normally have a certain level of experience and expertise. However, it has been proven that graduate students come across hurdles when trying to interact with open source software, often causing them to be discouraged (Cereceda, 2020). If they are still coming across these problems, there is support that these situations impact programmers of all experience levels. These issues can range from setting up a proper workspace, coming across unclear code comments, or documentation that is outdated (Cereceda, 2020). The commonness of outdated documentation causes problems for newcomers because they reference an incorrect version of how the code works, causing errors and confusion. Outdated documentation eventually becomes useless. Having outdated documentation will result in having no useful documentation, which is the equivalent to having no documentation. The lack of documentation causes programmers distress because they are unable to edit pre-existing code because they do not know what it does. Additionally, if the original code is difficult to comprehend and edit, this also causes problems in the infrastructure of the code because developers do not fully understand what the code does. As the development community grows, more code will be written. However, there is not enough documentation being written to support these codebases and constant updates. This issue in the lack of properly updated documentation needs to be addressed to maintain a sustainable code base within the development community.

Since the rise of the internet in the late 1990s, it has been well identified that documentation is necessary for the success and growth of software development. The most common form of documentation was the use of style guides, which created a basis for the
development of a certain software or program, and provided the principles a product or service was built on (Ohnemus, 1997). Ohnemus (1997) attributes the success and rapid boom of the internet, which was referred to as the World Wide Web, to the availability of open source software and existing documentation that was available at the time. While the available software was limited at the time, there was informative documentation of it, allowing developers to create more advanced and sophisticated programs we use today.

Since then, open source software has only grown, becoming the primary method of development within technical communities. Lenarduzzi (2020) states that one of the biggest reasons open source software is not even more widespread is the lack of updated and reasonably detailed documentation. There is a large population of developers who approach open source software with the intent to contribute, only to be frustrated by confusion (Lenarduzzi, 2020). Lenarduzzi (2020) also highlights that there are a multitude of published papers from the last twenty years about open source software adaptation concerns, confirming the unsustainability of current practices. Documentation is among the top five concerns listed in these published studies (Lenarduzzi, 2020). While the primary factors stakeholders value differ depending on their stance like financials versus developers, because the adaptation of open source software is still so limited from the difficulty of assessing its quality, it provides more support in how documentation can help to combat this assessment.

Open source software has grown significantly over the years. Goh (2006) identified one of the largest downfalls of open source software to be the lack of formal support such as proper
documentation. She discusses how this support is often provided through the usage of forums and mailing lists, practices which are not verified or sustainable due to the lack of organization and structure these methods have (Goh, 2006). While forums provide a means to an end, it can be difficult to hold these sources accountable, because anyone can add to these forums, and most responses are to specific issues that arise. This can cause problems among development communities because unless an issue is exactly the same, the response will not be useful to other developers having similar issues.

One of the goals of open source software is for new contributors to add to projects (Hull, 2021). However, research like Hull’s article found that open source software provides a multitude of barriers for newcomers to cross, causing developers to be discouraged from joining projects. This systematic literature review identifies the ways in which technical writers can alleviate these burdens, and how contributing to open source software as part of technical communication could be beneficial for students. Yet again, this article states that there is a large disconnect between source code in open source software projects and available documentation. Among these issues include the lack of detail in documentation, outdated documentation, and the overload of information by providing an explanation that is overwhelming and difficult to understand. The article identified about 70% of barriers were rooted in the issue of improper documentation practices (Hull, 2021). It also identified how technical communicators could play a larger role in these projects to provide more support. For example, technical writers could maintain and update README.texts, which are introductory files about a project, comment on
source code during updates, and create accessible documentation about the project. This would alleviate a significant amount of issues newcomers face when joining projects. Additionally, it introduces how technical writers should be utilized in these projects and their importance to helping maintain sustainable codebases with relevant documentation for future developers.

In Burn’s reintroduction of the hypergram, he highlights how unreadable source code causes problems for future developers. Since documentation practices like style guides have been introduced and utilized, researchers have tried to identify different and more effective ways to read and document code (Burns, 2021). The hypergram is an illustration responsive to user interaction by providing more information when part of an image is selected (Burns, 2021). Interchangeable with image map, smart drawing, and knowledge diagram, the idea was originated by Ted Burns to present concepts in one or many forms for technical communication. Hypergrams can provide the ability for technical communicators to create documentation that is more responsive, mobile friendly and better integrated into the program (Burns, 2021).

On a separate note, some of the most inexperienced of programmers would be those in college. While individuals are learning the basics of programming and trying to understand the concepts of coding languages, there are barriers that they have to overcome to try and contribute to any open source software projects (Cereceda, 2020). Because open source software projects are public domain and available to anyone, this makes it easy for students to contribute and collaborate with these projects. Cereceda (2020) does a deep analysis of the difficulties with interacting with open source software and the methods students attempt to overcome them. When
identifying methods of seeking help and the barriers of open source software, Cereceda (2020) discusses the unfamiliarity between interpreting error messages, unfamiliar functions, and uncertainty of previously written code without documentation. It is stated that almost all students come across the problem of open source software documentation, ranging from none of the documentation being useful enough to solve their problems to not being able to understand commentary because the documentation did not make sense. Cereceda (2020) states that 93% of respondents reported problems with incomplete, outdated, or confusing documentation. This is an incredibly high percentage of participants who faced issues due to the lack of documentation, proving that this issue is prominent, especially among inexperienced developers. They discuss the usage of forums to get guidance for their problems, like the frequently visited StackOverflow (Cereceda, 2020). However, they mention that forum websites may provide a useful amount of information, but the answers provide too high of a scope for students to follow, causing them to be just as confused and unable to make progress with issues they come across (Cereceda, 2020).

Due to the lack of formal documentation of open source software, the usage of forums is very popular among the development community. These are places for developers to post code and issues they are facing in their programs. They open it up to community members to help provide feedback or solutions. While these forums are helpful, there are still underlying problems with these methods that remain unsustainable. Berger (2020) analyzes how technical communication in addressing user problems is situated within use cases that users themselves aren’t aware of (Berger, 2020). He addresses how the usage of forums are slowly fading out,
because conversations are moving toward large corporate messaging services, like Slack and Microsoft Teams (Berger, 2020). However, he states that these methods are not effective and causes users difficulty in knowing what state of mind they are in, impacting the ability to properly document their code (Berger, 2020). Because all the forums and messaging services are done in real time, the information provided is only relevant to specific situations, which eventually becomes unorganized and difficult to navigate to find relevant information. While Burns does argue that a forum provides more individualized support than a written manual, he enforces the idea that written documentation is necessary for users to have something to work with and a lack of proper documentation causes there to be more problems.

Over time, the methods in which open source software documentation is recorded has emerged. As previously discussed, online forums and messaging services provide assistance for many developers. Forums are more frequent for open source software projects, while in companies, developers might share ideas and issues through messaging services. Swarts (2015) states that although online forums can assist and be more effective by having a live troubleshooting session, these circumstances are unsustainable and he explains the reasoning for why documentation exists in the first place. Swarts’ study looks into how certain documentation practices can prove to be unhelpful. For example, going through wordy blocks of text creates further confusion for users, causing them to lean more towards conversational avenues like forums, where there is more of a dialog than a manual language (Swarts, 2015). However, he identifies that forums provide an interesting method for documentation, but only allows for a
place with specific questions. Therefore, in this article, they explain how documentation practices need to be more readable and consistently updated to provide usage to users.

The majority of Big Tech companies utilize a style guide of some regard to provide a standard for their codebase and establish programming strategies for their specific company. Companies vary in how they wish to structure and organize their style guides. There is a large variety in style guides utilized by large companies versus small startups, as it normally is dependent on the company’s mission statement and culture. While larger companies might have more complex and detailed codebases, their style guides should reflect this and remain updated to ensure that there is consistent documentation for any changes.

For example, Google’s style guide provides an abundance of information and is easily accessible through their website. However, their webpage is overwhelming with a dense sidebar that can cause the user to lose track of where they were originally. Google’s style guide consists of many links, causing the user to switch web pages. While the web pages are organized in a structured and readable way, the variety of webpages and switching from screen to screen can be confusing and cause users to be overwhelmed by going through different pages to find an answer. This style guide also lacks specific coding examples for programmers to reference their own code and understand what they did wrong. The majority of such style guides provide paragraph explanations on a certain topic, or references to an external link to another webpage, causing the user to jump to a new tab, which creates more confusion. While there is a bountiful amount of information collected in a style guide like Google’s, it can be ineffective. It causes a
scavenger hunt for the user to find the information they are looking for or to see an example that allows programmers to compare it to their own.

On the other hand, another Big Tech company that utilizes a very effective and navigable style guide is AirBnB. AirBnB’s style guide is one long webpage, which allows for simplicity and ease of use. Additionally, their style guide provides code examples, allowing programmers to easily reference and understand their own code and problems. This style guide also has a table of contents. When the user selects a topic, they are easily brought to that section of the webpage, but they remain on the same initial website to prevent them from getting lost. The style guide is simple and easily navigable, and has example code blocks with explanations. This style guide is also a webpage like Google’s. However, AirBnB’s organization of material and easily accessible examples improves the effectiveness of their style guide dramatically.

Through these observations, there is a common trend that occurs during documentation in open source development. Frequently in the beginning, when a project is in the first stage of development, initial documentation is created, explaining original steps and processes. However, because beginning steps are oftentimes very premature, programmers do not provide enough documentation explaining the work, or fail to explain it well enough for a new developer to understand. This lack of detail in documentation creates difficulty for incoming developers to understand the codebase. Additionally, premature code is and should be adjusted, changed, and improved upon as development continues. Therefore, when there are changes to the code, there should also be edits to the respective documentation. However, in the middle of development,
when these changes occur, it is quite common that the relevant changes aren’t made, or that deprecated documentation is not taken down, which can cause great confusion for developers. The result of this leads to a negative feedback loop. Because the old documentation is not being updated closely enough, the existing documentation becomes useless, causing there to be no helpful documentation for developers to use or reference. This is a frequent issue that arises in open source software that can discourage new developers from joining and contributing to new projects.

In order to better understand these issues, observation should be done between software developers, technical writers, and their companies. Software developers come across documentation issues and it directly impacts their work. They can identify the current documentation practices and frequency in how often there is proper documentation or appropriate comments for their code. Additionally, they can record how frequently they come into documentation errors allowing them to personally understand how these issues impact them and help identify solutions. Technical writers are also an important role because it is their job to write the documentation and create resources like style guides. It would be interesting to understand how frequently they update previous documentation in comparison to writing new documentation for separate projects. It would also be interesting to identify their workload and understand if they spend more time on a specific team or constantly writing new content for different teams. In order to write technical documentation, there also has to be some level of programming experience, and it is notable to identify how much most technical writers require.
Lastly, it would be helpful to understand how companies prioritize the documentation of their codebase. Companies can observe how many technical writers are hired to provide documentation for all of the software engineers. There should be a ratio to identify the difference between software engineers and technical writers, and if this ratio is suitable enough to maintain proper and updated documentation practices. To better understand motivations of these two roles, observing the salary differences between the two positions and finding how it might play a role in open source software’s documentation practices would also be interesting.

From the observed data and literature available, I have a proposed hypothesis to explain why open source software is struggling with maintaining proper documentation. The underlying issue would relate to the amount of technical writers adding to available documentation. Technical writers are not prioritized as heavily as software engineers, and therefore, the demand for documentation cannot keep up with the influx of new and updated code. Then, available technical writers are only able to maintain so much documentation without enough support. Understaffed writers will either fall behind and cause documentation to lack in areas, or they are prioritizing creating new documentation for new projects, causing old documentation to fall at the wayside. Additionally, the ability to write effective documentation requires a significant amount of technical knowledge and expertise. If a technical writer has the skills of a software engineer, but a software engineer’s salary is higher than a technical writer’s, the motivations to become a technical writer decrease even more, making the amount of them in the industry more scarce, contributing back to the original problem. Because of this lack of motivation for
individuals to become technical writers, there is a large gap between the code being created and
the supporting documentation being written. This is creating a large neglect on the
documentation practices, which is resulting in unsustainable practices for future developers.
While these problems persist throughout the development community, it will lead to future
frustrations for developers. This brings the proposal of how to create effective documentation
practices in open source software to ensure the success of the software and the written
documentation.

To create an effective style guide for Tsuboniwa: A Garden Dream, I would recommend
developing a document or webpage that provides a table of contents of all the topics covered
throughout the project. This includes but is not limited to the flocking algorithm, FlockBox,
Unity interface, and music implementation, among others. After observing the C# files and
summarizing what each file consists of, I would create a cohesive table of contents organized by
the topic, organization hierarchy, and type of information (code, interface, setup, etc). For each
section, I would write an excerpt of necessary information needed to understand the matter, and
to provide examples of the correct practices. I would also ensure that there are comments in these
respective files to ensure that there is relevant information needed at the locations in which it is
in relation to. This can include frequent commenting in the code and references to more in depth
sections of the style guide when needed. A style guide for this project would allow for the future
developers to have a clear understanding of the completed work and how it was made. To ensure
that the style guide remains relevant for incoming developers, it is vital that this documentation
remains editable so that it can be revised and revisited as the code gets adjusted. Through a collection of personal experience, literature review, and industry examples, the need for proper and more effective documentation practices in open source software is necessary to ensure a sustainable and manageable code source for future developers to build upon.
References


