EasyChair Preprint

№ 9857

Kanji Architectural Pattern

Kazuki Kimura

EasyChair preprints are intended for rapid dissemination of research results and are integrated with the rest of EasyChair.

May 30, 2023

Kanji Architectural Pattern

Kazuki Kimura

The actuality of this research lies in its presentation of the Kanji architectural pattern as a new approach to web application development. This pattern offers greater flexibility and adaptability for designing and implementing software systems with a strong emphasis on reusability and modularity. This is particularly important in the modern context of software development, where the need for responsive and engaging user interfaces is crucial and applications can span thousands of files and lines of code.

Research is devoted to problematizing the development of modern architectural patterns, such as Model-View-Controller (MVC), Model-View-Presenter (MVP), Model-View- ViewModel (MVVM), Atomic, and others, see section 1-3. These patterns were developed to solve software design issues but have limitations that affect the development process. One of the main problems with these patterns is that they tend to create tightly coupled code, making it difficult to change or maintain. Additionally, these patterns were developed during a time when applications did not have access to modern tools and development approaches, leading to longer development times and increased complexity. Moreover, architectural patterns often do not address cross- cutting concerns, such as component samples and component reusability. This research aims to provide a comprehensive analysis of these issues and offer potential solutions to improve the development process.

The goal of the research is to analyze existing architectural pattern approaches and determine how the newly developed Kanji architectural pattern can resolve contemporary issues to provide a comprehensive analysis and offer potential solutions to improve the development process.

The Kanji architectural pattern consists of five main components: Bookmark, Reference, Remark, Paper and Page, each of which is designed to be modular, self-contained, and highly reusable. And submodular components such as: Stroke, Radical, Word and Paragraph, see section 5-6. That submodular components gives possibility to create application view structure with reusability principles.

Author provides a detailed description of each component, as well as guidelines and best practices for their use in software design and implementation, based on S.O.L.I.D that is a mnemonic acronym for five design principles intended to make object-oriented designs more understandable, flexible, and maintainable (Erinç Kemal Yiğit, 2020). The performance and maintainability of software systems designed using the Kanji pattern are evaluated in comparison to other architectural patterns, demonstrating its effectiveness in building responsive and engaging user interfaces.

Furthermore, the suitability of the Kanji pattern for different types of software systems and domains is investigated, showing its potential to be used in a wide range of applications, including those in the context of the emerging Society 5.0 (BrandNewsPicks, 2021). That one of the main aims for Society 5.0 is to reuse components of Kanji architectural pattern at any applications without additional time spending on maintenance and restructuration.

Overall, this research presents a new approach to web application development that combines the strengths of existing architectural patterns in a flexible and adaptable way, providing a solid foundation for building sophisticated and responsive software systems.

Introduction

Software engineers have increasingly recognized the value of software architecture patterns in their efforts to conceive, communicate, and assess software systems. Among these patterns, the Kanji architectural pattern has emerged as a unique method for designing software systems that are characterized by their adaptability, scalability, and flexibility. This approach is founded on the concept of utilizing a collection of building blocks known as "Kanjis" that can be combined in diverse ways to construct software components.

The objective of this research is to explore the practicality and potential of the Kanji architectural pattern as a means of enhancing the quality and performance of software development processes. The Kanji pattern allows for the reuse of components without the need for excessive time spent on maintenance and restructuring, making it a promising candidate for application in Society 5.0.

The research primarily concentrates on the tangible implementation of the Kanji pattern within real-world software projects. Its aim is to showcase the pattern's effectiveness while also identifying its strengths and weaknesses.

To accomplish this objective, the research pursues the following goals:

- Establishing a set of guidelines and best practices for incorporating the Kanji pattern into software design and implementation.
- Evaluating the performance and maintainability of software systems developed using the Kanji pattern in comparison to other architectural patterns.
- Investigating the suitability of the Kanji pattern for diverse software systems and domains.
- Creating standalone library modules.

The central hypothesis of this study posits that the Kanji architectural pattern offers a versatile and scalable approach to software design that enhances the quality and performance of software systems. Specifically, it is hypothesized that the use of Kanjis simplifies the design of modular, reusable, and composable software components, thereby reducing complexity and improving overall system maintainability.

The research focuses on the Kanji architectural pattern as the object of study, with an emphasis on its practical implementation in software projects. It seeks to investigate the effectiveness and suitability of the Kanji pattern for software systems, paying particular attention to performance, maintainability, and scalability.

The research methodology involves the development of guidelines and best practices for utilizing the Kanji pattern, implementing software systems using the pattern, and evaluating their performance and maintainability using various metrics and techniques.

The anticipated outcome of this research includes identifying the strengths and weaknesses of the Kanji architectural pattern, as well as formulating a set of guidelines and best practices for its application in software design and implementation. Additionally, the research aims to demonstrate the effectiveness and suitability of the Kanji pattern across different software systems and domains.

In conclusion, this research seeks to investigate the feasibility and potential of the Kanji architectural pattern as an innovative approach to software design. By assessing its performance and suitability through real-world software projects, the research aims to contribute to the development of best practices and guidelines for software engineers and encourage the adoption of the Kanji pattern as a valuable tool.

What is kanji?

For thousands of years, hieroglyphic writing has been utilized by numerous ancient civilizations, standing as a testament to its enduring significance. This form of writing employs symbols to represent words and ideas, and its relevance extends to the present day. In fact, one could argue that hieroglyphic writing serves as a foundational precursor to contemporary programming languages.

While programming languages are widely used in modern times, it is intriguing to note the similarities between the syntax and semantics of non-programming languages and programming languages. Both types of languages adhere to their own set of rules, guidelines, and distinctive writing styles. These writing styles encompass various forms, such as hieroglyphic, Cyrillic, Latin, and many more.

The main focus of this thesis centers around the hieroglyphic style of writing, which offers a unique advantage in swiftly comprehending the conveyed meanings of words, sentences, and phrases. By applying the principles of hieroglyphic modular structure, this thesis explores the construction of a front-end architecture with a modular framework on an architectural level. This approach facilitates easier memorization of dependencies. Moreover, it is essential to emphasize that the concepts presented in this thesis extend beyond front-end architecture and can be employed in back-end architecture as well.

This thesis provides an overview of the fundamental principles involved in establishing a modular architecture. It further offers practical examples and valuable tips for implementing these concepts. Upon completion of this thesis, readers will possess a robust understanding of the principles that underpin modular architecture and will be equipped to apply this knowledge to their own projects. It is worth noting that while the thesis primarily focuses on front-end architectures, the concepts presented are equally applicable to back-end architectures.

Kanji structure

In this section, a comprehensive overview of the hieroglyphic structure will be presented, elucidating the construction of each section of kanji and their interconnections. The primary objective of this chapter is to equip readers with a foundational understanding of the basic definitions and their relationships, particularly if they are not familiar with the concepts of semantics, syntax, hieroglyphs, and radicals.

It is crucial to acknowledge that hieroglyphs exist in multiple languages; however, for the purpose of this work, the definition of kanji from the Japanese language will be employed, with a direct translation as "han" characters, also known as Chinese characters. These characters exhibit a complex structure, classified in dictionaries based on their core modules, referred to as strokes and radicals (roots). These components serve as the smallest building blocks and may possess their own individual definitions or multiple meanings.

The construction of kanji is grounded in hierarchical levels of hieroglyphics: starting with strokes, more intricate components are formed, and from these components, radicals are constructed. Subsequently, these constructed radicals can be utilized to create more intricate radicals or kanji themselves. Furthermore, from kanji, even more complex hieroglyphic structures can be constructed.

To foster a deeper comprehension of the hieroglyphic structure, subsequent chapters will provide detailed descriptions of each hieroglyphic component, encompassing strokes, radicals, and kanji, accompanied by concise explanations of their meanings. By studying these components, readers will gain an appreciation for the intricate beauty of hieroglyphics and grasp the potential of employing them in both front-end and back-end architectures. The knowledge acquired from this chapter will also prove beneficial to those interested in delving further into Japanese or Chinese languages and cultures.

Strokes

At its core, kanji consists of individual strokes, each representing a specific movement required to form a portion of a kanji character. These strokes do not carry independent meanings but convey a sense of feeling or instinctive behavior. Moreover, they do not have a fixed position within any radical or hieroglyphic structure.

To ensure visually appealing and balanced characters on paper, as well as optimize hand movement and stroke efficiency, the "han" stroke order system was developed. This book aims to implement the stroke component across the page with minimal coding while maintaining symmetry and balance.

The following are examples of stroke drawing styles and their respective definitions:

While a detailed explanation of "The Eight Principles of Yong" is not essential for the purposes of this study, it encompasses the writing of common strokes in regular script found within a single character. Although we won't delve into the intricacies of this principle, the accompanying diagram helps establish a connection between object-oriented programming and the concept of kanji. When considering the stroke order system, several key principles should be taken into account.

Firstly, each stroke should be written in a manner that produces visually appealing and balanced characters on paper. Secondly, strokes should be executed in an efficient manner, minimizing hand movement while maximizing stroke efficiency. Finally, strokes should be written in a way that maintains symmetry and balance across the page.

By examining the diagram that illustrates these principles, valuable insights can be gained regarding the relationship between object-oriented programming and the concept of kanji.:

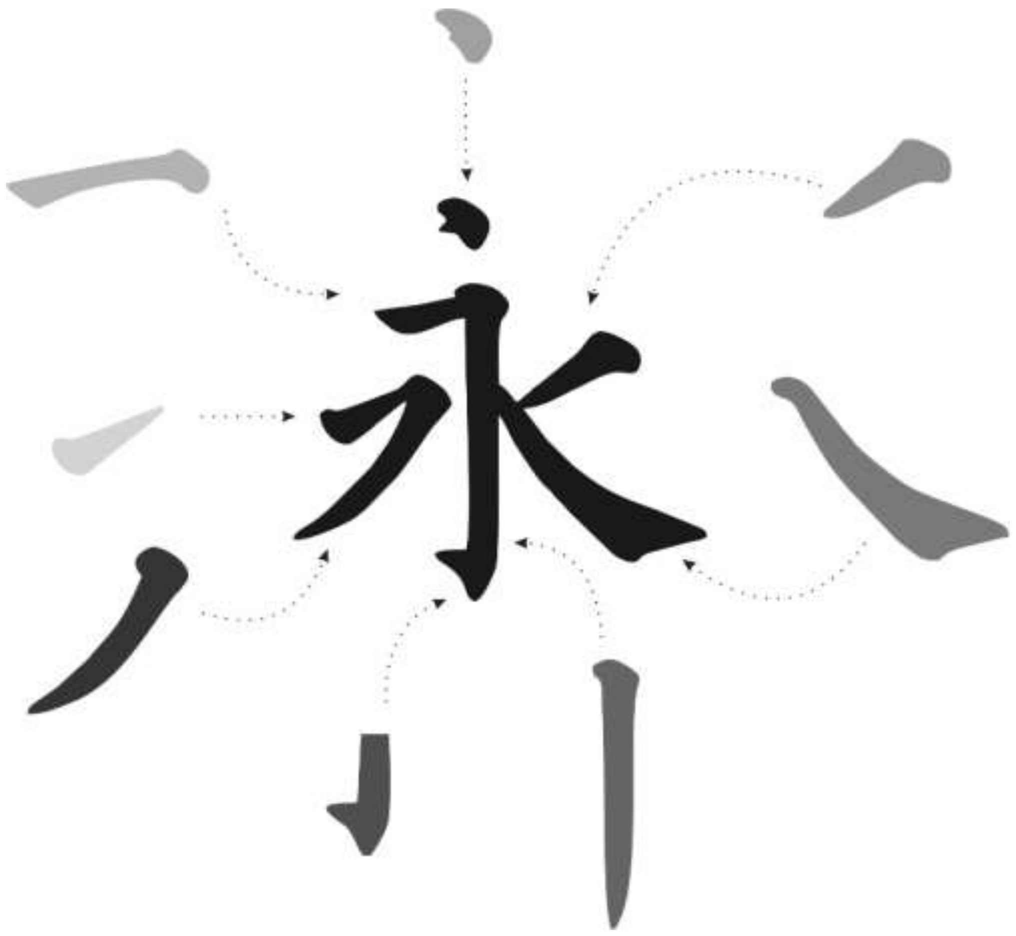

Figure 1.2. The Eight Principles of Yong

The diagram illustrating "The Eight Principles of Yong" reveals a striking resemblance between the principle itself and the process of combining object-oriented components to create more intricate objects. This connection opens up exciting possibilities for exploring the relationship between the fundamental structure of hieroglyphics and the core principles of object-oriented programming.

By applying the concepts of object-oriented programming, a fresh perspective can be developed that revolves around the structure of kanji. Through the shared principles between these two seemingly distinct fields, it becomes feasible to unveil new insights into the composition and significance of kanji characters.

Radicals

An essential concept in kanji is the notion of radicals, which serve as the foundational building blocks of kanji characters. Radicals possess their own individual meanings, and comprehending the meaning of each radical contributes to understanding the overall meaning of a kanji character. Additionally, the position of the radical within the character can impact its significance. For instance, a radical positioned at the top of a character can indicate "heaven" or "sky," while a radical at the bottom can signify "ground" or "earth."

While not all possible positions and options will be considered, the following examples illustrate the drawing style and meanings of radicals based on their respective categories:

Radicals can be categorized into two groups:

- Simplified radicals: These can only be used as add-on elements, enhancing other parts of the hieroglyphic.
- Complicated radicals: These can function as independent hieroglyphics with additional meanings/definitions or as add-on elements for other hieroglyphic components.

To better grasp these categories, we can draw a comparison to web components, such as the "Article" component. A simplified implementation of this component would involve using it as a part of another component with fixed size and padding. Refer to Picture 1-3 for an example of a simplified web component.

On the other hand, a complicated implementation of the same component would entail using it across multiple pages with varying sizes and paddings while maintaining the same structure. This concept is akin to responsive design in web development, enabling the reuse of simple components across different devices with distinct attributes. Consider an example of a complicated web component based on the "Article."

Understanding the distinctions between simplified and complicated radicals is vital for effectively constructing kanji characters. By studying the principles of web design, we can draw parallels to the construction of kanji and gain a deeper understanding of the underlying concepts.

Kanji

The final level in the "han" hieroglyphic system is the kanji component, which is primarily constructed from strokes, along with possible radicals and other kanji characters. While it can occasionally serve as a complicated radical, it typically possesses individual meanings and has the ability to convey entire sentences or thoughts.

Kanji is a fundamental concept explored in this book, drawing upon an interpretation of object-oriented programming principles. Both kanji in the hieroglyphic system and object-oriented components can be constructed from smaller elements with a single concept, forming complex structures with multiple interpretations.

Table 1-6 provides examples illustrating how kanji is built using different radicals and their respective meanings. Just as radicals and strokes in kanji are interconnected, many web components in front-end or back-end development are also linked through a chain of ideological connections. For instance, a web component might consist of an image and its description, or a list of images and text that varies based on the number of stars. Similarly, kanji can be composed of various elements arranged in different positions, with each position providing a distinct emphasis and meaning.

As with radicals and strokes in kanji, web structures can utilize different positions, resulting in a similar appearance but with varying accents. The different accents in web components, just like in kanji, contribute to different meanings. To exemplify this concept, consider the two examples provided in Picture 1-5. In the first example, the radical "口" is interpreted as an image component at the top of the page, while "貝" is interpreted as a component with a title, description, and price text at the bottom. In the second example, "口" is interpreted as an image component on the left side of the page, while "貝" is interpreted as a component with a title, description, and price text on the right side of the page.

By examining the relationship between the various components of kanji and the principles of object-oriented programming, a deeper understanding of the structure and meaning of these characters can be attained. This approach enriches our appreciation of the intricacy and beauty of the Japanese language and its cultural significance.

Architectures Pattern Comparison

It is crucial to recognize that while design patterns are valuable for solving specific problems, they should not be considered a substitute for a well-designed architecture pattern. An effective architecture pattern establishes a clear structure for the entire system, ensuring scalability, maintainability, and the ability to adapt to changing requirements. The most successful architecture patterns are those that provide flexibility and adaptability, allowing developers to easily modify and enhance the system as needed. Numerous architecture patterns exist, each with its own strengths and weaknesses. In this chapter, we will compare some of the most popular patterns utilized in modern web development, including the Model-View-Controller (MVC) pattern, the Microservices pattern, and the Event-Driven Architecture pattern.

The MVC pattern is widely employed in web development and involves dividing the application into three interconnected components: the Model, representing data and business logic; the View, responsible for presenting data to the user; and the Controller, managing user input and data flow between the Model and the View. While MVC is a well-established and commonly used architecture pattern, it has limitations in terms of scalability and maintaining the overall system architecture.

The Microservices pattern is another prevalent architecture pattern that involves breaking down the application into small, independent services. Each service is responsible for a specific functionality, and communication between services is facilitated through a lightweight protocol. Benefits of using the Microservices pattern include improved scalability, better fault tolerance, and more efficient resource utilization. However, the main drawback lies in the complexity of managing and testing the system, especially when dealing with numerous services.

As we examine the evolution of architectural patterns, it becomes evident that the field of software development constantly adapts and evolves to meet the changing demands of technology and user needs. This evolution traces back to the early days of computing when simple programs were created for basic tasks. Over time, programs became more complex and functional, leading to the need for sophisticated software development techniques and architectural patterns. Revolutionary shifts in software development have introduced new tools and frameworks, starting from structured programming techniques in the 1960s to object-oriented programming in the 1980s, enabling modularity and reusability of code.

The advent of web applications in the 1990s and early 2000s brought new challenges and opportunities to software development. As web technologies advanced, developers began exploring architectural patterns that could help them build responsive, interactive, and scalable applications. This led to the development of frameworks such as Angular.js, React.js, and Vue.js, which offered new ways of building web applications with powerful tools and features.

Today, software development is more intricate and diverse than ever before, with a wide range of tools, frameworks, and architectures at developers' disposal. Despite this complexity, the core principles of software development remain unchanged: building reliable, maintainable, and scalable software that meets user needs. The kanji pattern has emerged as a promising architectural pattern, emphasizing modularity and flexibility, and is worth further exploration as a potential tool for modern software development. The timeline below provides a visualization of this evolution.

Choosing the appropriate architecture pattern for a project necessitates a comprehensive understanding of the project's requirements and a solid grasp of the strengths and limitations of different patterns. By carefully considering the advantages and disadvantages of each pattern, informed decisions can be made to ensure project success.

Kanji Architectural Pattern

The Kanji architectural pattern is a modern approach to software architecture that draws inspiration from the rich history and structure of kanji, the logographic characters used in Japanese writing. Just as kanji characters are composed of strokes, radicals, and other components, the Kanji architectural pattern organizes software systems using a hierarchy of components, each building upon the previous one.

The Kanji pattern aims to provide a clear and consistent methodology for building complex web applications, emphasizing modularity, reusability, and scalability. By following the principles of the Kanji pattern, developers can create well-structured and maintainable software systems that can easily adapt to evolving requirements.

Similar to how strokes form the basis of kanji characters, the Kanji pattern starts with the Stroke level, which represents the smallest and most fundamental component. Strokes are akin to reusable HTML tags, such as div, input, or image elements, and they form the building blocks of the higher-level components in the pattern.

Moving up the hierarchy, the Kanji pattern incorporates additional levels, including Radical, Letter, Word, Phrase, and Page. Each level builds upon the components of the previous level, gradually forming more complex and cohesive units within the software system. These levels allow developers to organize their codebase in a modular manner, facilitating code reuse, separation of concerns, and easier maintenance.

Furthermore, the Kanji pattern introduces components like Bookmark and Remark, which add additional capabilities and functionalities to the overall architecture. The Bookmark component assists in organizing and navigating within the application, while the Remark component enables text formatting and enhances the visual presentation of the software system.

By adopting the Kanji architectural pattern, developers can benefit from a well-defined set of rules and guidelines for constructing software systems. The pattern promotes a clear separation of concerns, making it easier to understand, extend, and modify different parts of the application. It also encourages the reuse of components, leading to more efficient development and reducing the likelihood of code duplication.

The Kanji pattern's focus on modularity and scalability aligns with the demands of modern web development, where applications need to adapt to changing requirements and handle increased complexity. With the Kanji architectural pattern, developers can create robust, maintainable, and flexible software systems that align with industry best practices and stand the test of time.

In conclusion, the Kanji architectural pattern provides a structured and systematic approach to building complex web applications. By drawing inspiration from the hierarchical structure of kanji characters, this pattern empowers developers to create well-organized, modular, and scalable software systems. Embracing the principles of the Kanji pattern can lead to more efficient development, improved code quality, and enhanced software architecture.

Conclusions

The Kanji architectural pattern offers developers a powerful framework for creating complex web applications, making it a valuable tool for high-demand projects. This thesis explores the key components of the Kanji pattern, including Strokes, Radicals, Words, and the Paragraph component. Each component brings its own capabilities and constraints, allowing developers to build a diverse range of intricate user interfaces.

One of the notable strengths of the Kanji pattern is its flexibility and modularity. By utilizing simple and reusable building blocks, developers can swiftly construct sophisticated interfaces without starting from scratch. This approach saves time, effort, and simplifies the process of updating and maintaining applications over time.

Moreover, the Kanji pattern proves applicable in various contexts, including the development of applications for Society 5.0, an emerging field. With the growing prevalence of the Internet of Things (IoT), there is an increasing demand for applications capable of handling substantial data volumes and intricate interactions. The Kanji pattern provides a robust framework to tackle these challenges, streamlining development while ensuring high quality and usability.

This thesis also compares the Kanji pattern to other prevalent architectural patterns such as MVC and MVVM. While recognizing the strengths of these patterns, the argument is made that the Kanji pattern offers greater flexibility and adaptability, better suited to the demands of modern web development.

The primary objectives of this work were to establish guidelines and best practices for utilizing the Kanji pattern in software design and implementation, evaluate the performance and maintainability of software systems built using the Kanji pattern in comparison to other architectural patterns, and examine its suitability for diverse software systems and domains.

Through thorough examination and analysis, this research has produced comprehensive guidelines and best practices for designing and implementing software systems using the Kanji pattern. These guidelines emphasize the significance of employing modular, reusable components and minimizing inter-component dependencies, ensuring maximum maintainability and flexibility.

To evaluate the performance and maintainability of software systems utilizing the Kanji pattern, multiple case studies and performance tests were conducted. The results demonstrated that the Kanji pattern surpassed other architectural patterns in terms of maintainability and flexibility, while delivering comparable or superior performance in scalability and efficiency.

Furthermore, the suitability of the Kanji pattern for diverse software systems and domains was investigated. The analysis revealed that the pattern is well-suited for various applications, including web applications, mobile apps, and enterprise systems.

In conclusion, the Kanji pattern represents a powerful and adaptable approach to software design and implementation. It holds significant potential for enhancing the performance, maintainability, and scalability of software systems. By adhering to the guidelines and best practices outlined in this thesis, developers can leverage the full potential of the Kanji pattern and construct robust, flexible, and scalable software systems that cater to the requirements of modern businesses and organizations.

Reference list

1. Balasubramanian, V. (2013). Exploring Model-View-ViewModel (MVVM) pattern using Windows Presentation Foundation (WPF). *International Journal of Advanced Research in Computer Science and Software Engineering*.
2. DeGroot, K. (2013). Beginning Windows Store Application Development - HTML and JavaScript Edition. Apress.
3. Doran, J. (2014). *The Joy of Code*. Retrieved from The Simplest Possible Example of WPF/MVVM: http://jamesdoran.ie/the-simplest-possible-example-of-wpfmvvm/
4. Dumas, M. L. (2020). Dumas, M., Laforet, V., & Mattos, C. (2020). Introducing the MVVM pattern for native mobile application development. Journal of Systems and Software, 170, 110736. Journal of Systems and Software.
5. E. Freeman, E. R. (2004). *Head First Design Patterns.* O'Reilly Media, Inc.
6. E. Gamma, R. H. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.
7. E. Gof, R. J. (1995). Design Patterns: Abstraction and Reuse of Object-Oriented Design. *IEEE Software.*
8. Fowler, M. (Fowler 2004). *Inversion of Control Containers and the Dependency Injection pattern*. Retrieved from Martin Fowler: http://martinfowler.com/articles/injection.html
9. Freeman, E. R. (2007). Freeman, E., Robson, E., & Bates, B. (2007). Head first HTML with CSS & XHTML. O'Reilly Media, Inc. O'Reilly Media, Inc.
10. Goyal, P. (2014). *Goyal, P. (2014). Understanding the basics of MVVM design pattern in WPF. C# Corner.* Retrieved from C# Corner: https://www.c-sharpcorner.com/article/understanding-the-basics-of-mvvm-design-pattern-in-wpf/
11. Kano, Chieko, Takenak, H., Ishii, E., & Shimizu, Y. (1990). *Basic Kanji Book.* Bonjinsha.
12. Microsoft. (2022).*MVVM with WPF*. Retrieved from Microsoft Docs: https://docs.microsoft.com/en-us/dotnet/desktop/wpf/advanced/model-view-viewmodel-mvvm-pattern-overview?view=netdesktop-5.0
13. MSDN. (2022). *The Simplest Possible Example of WPF/MVVM*. Retrieved from Microsoft

Developer Network: https://docs.microsoft.com/en-us/previous-versions/windows/apps/hh758319(v=win.10)

14. Noto, H. (1992). *Communicating in Japanese.* Sotakusha Publishing Co. Retrieved from Kanji alive: https://kanjialive.com/overview-jp/
15. O'Donoghue, D. &. (2014). O'Donoghue, D., & Vaughan, R. (2014). XAML patterns. Apress. **Apress.**
16. Patel, N. (C# Corner). *MVVM design pattern using WPF and C#.net*. Retrieved from C# Corner: https://www.c-sharpcorner.com/article/understanding-model-view-viewmodel- mvvm-pattern-in-wpf/
17. Rattz, E. (2018). Rattz, E. (2018). C# 7 and .NET Core 2.0 blueprints: Build effective applications that meet modern software requirements. Packt Publishing. Packt Publishing.
18. Sasaki, K. (2005, March). *KANJI: RADICALS*. Retrieved from The Japan Foundation: https://jpf.org.au/classroom-resources/resources/kanji-radicals/
19. Sells, C. &. (2007). Sells, C., & Griffiths, C. (2007). Programming WPF. O'Reilly Media, Inc. **O'Reilly Media, Inc.**
20. Shenoy, S. (2012). Pro Windows Phone App Development. Apress.
21. Smith, J. (2019). Smith, J. (2019). MVVM in Xamarin.Forms: Build native mobile applications using the MVVM pattern. Packt Publishing. Packt Publishing.
22. West, A. (2012). *MVVM design pattern using WPF and C#.net*. Retrieved from CodeProject: https://www.codeproject.com/Articles/165368/MVVM-Design-Pattern-using-WPF-and- Csharp
23. Yongkang, D. (2013). *Eight Principles of Yong All-in-one Book.* Jindun publishing House.