REVERSE ENGINEERING MICROSERVICES

REVERSE ENGINEERING MICROSERVICES FOR ENHANCED INSIGHTS

BY

MUHAMMAD WAQAR UL HASSAN AWAN, M.Eng.

A Report

submitted to the Computing and Software

and the School of Graduate Studies

of McMaster University

in partial fulfilment of the requirements

for the degree of

Masters of Engineering

Masters of Engineering (2024) McMaster University

(Computing and Software) Hamilton, Ontario, Canada

TITLE:	Reverse Engineering Microservices for Enhanced Insights
AUTHOR:	Muhammad Waqar Ul Hassan Awan M.Eng. Computing and Software, McMaster University, Hamilton, Canada
SUPERVISOR:	Dr. Sébastien Mosser
NUMBER OF PAGES:	xiii, 33

Lay Abstract

A lay abstract of not more 150 words must be included explaining the key goals and contributions of the thesis in lay terms that is accessible to the general public.

Abstract

Abstract here (no more than 300 words)

Your Dedication

Optional second line

Acknowledgements

Acknowledgements go here.

Contents

List of Figures

List of Tables

Notation, Definitions, and Abbreviations

Notation

$A \leq B$	A is less than or equal to B

Definitions

Challenge	With respect to video games, a challenge is a set of goals presented to the player that they are tasks with completing; challenges can test a variety of player skills, including accuracy, logical reasoning, and creative problem solving

Abbreviations

SST	Single source of truth
UDS	Unified data source

RE	Reverse engineering
SRE	Software reverse engineering
DSL	Domain-specific language

Declaration of Academic Achievement

The student will declare his/her research contribution and, as appropriate, those of colleagues or other contributors to the contents of the thesis.

Chapter 1

Introduction

Every thesis needs an introductory chapter

While you're here, you need to go into `definitions.tex` to set all the information needed for the front matter (e.g. title, author) and page header/footer.

You will also find the School of Graduate Studies' preparation guide (August 2021) for theses and reports. I would give it a quick read so you know what's expected.

Chapter 2

Background

Every thesis needs an Background chapter

While you're here, you need to go into `definitions.tex` to set all the information needed for the front matter (e.g. title, author) and page header/footer.

You will also find the School of Graduate Studies' preparation guide (August 2021) for theses and reports. I would give it a quick read so you know what's expected.

Chapter 3

Vision and Technical Strategy

Developing a large software system is a complex and crucial process that requires careful planning and execution. A project of this scale involves many interconnected components, all of which must adhere to essential software development principles. The goal is not just to write code but to build a system that is reliable, maintainable, scalable, and efficient. Ensuring the system is free of bugs and capable of adapting to future needs is as important as the initial development itself. By following these key paradigms, developers can create software that meets high standards of quality and performance.

Since an already built large software system is quite complex, understanding this system requires certain strategies and tools. When a problem arises in complex software architectures, such as microservices or service-oriented architectures, resolving it often demands significant resources. These architectures consist of numerous interconnected components, and identifying the root cause of an issue can be challenging. The process may involve extensive debugging, analyzing logs, coordinating between multiple teams, and sometimes even re-evaluating design decisions. This can result

in considerable time, effort, and cost being spent to restore functionality and ensure the system operates smoothly. (Folmer et al., 2005) discusses the usability issues in software systems post-development and highlights that they require significant architectural changes. In order to deal with such architectural changes, an understanding of the whole system is required, and if the system is large enough, major resources are spent fixing even minor issues.

(Canfora and Cimitile, 2001) states that several surveys indicate that software maintenance consumes 60% to 80% of the total life cycle costs. Also, the maintenance costs are largely due to enhancement (often 75-80%), rather than corrections. To address these challenges, there is a growing need for tools that can assist in resolving bugs and reducing maintenance overhead. Such tools should be capable of reverse engineering software systems to provide a clear and comprehensive view of the architecture. By highlighting the key components and their interactions, these tools make it easier for developers to understand the system, identify issues, and perform necessary tasks efficiently. This not only simplifies debugging but also enhances the overall maintainability of the software, ensuring smoother operation and reduced downtime.

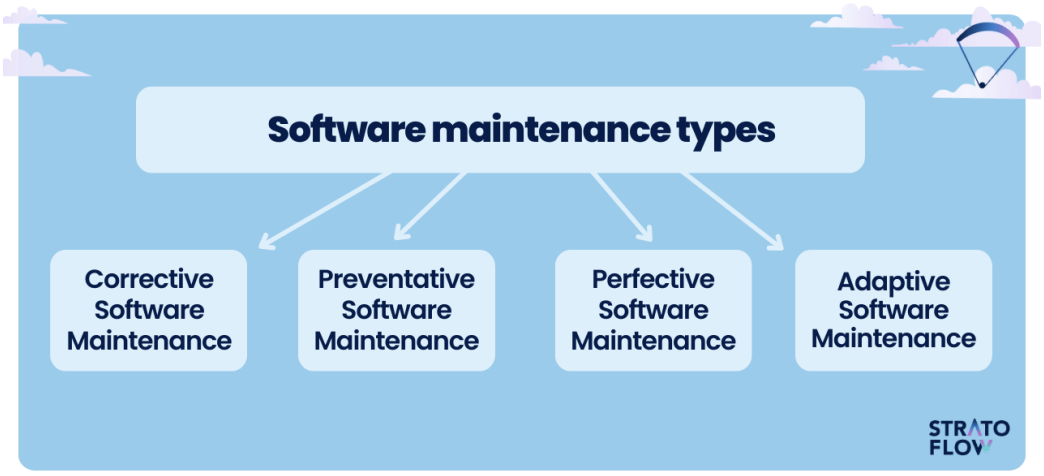

Figure 3.1: Software Maintenance Types (adapted from Stratoflow (2025))

In this chapter, we will explore the necessity of a framework for software analysis and its role in addressing critical challenges in modern software systems. We will outline the overall vision of the framework, highlighting its unique features and how it stands apart from existing solutions. Finally, we will discuss the technical strategy employed to guarantee the framework's comprehensive functionality, we will delve into the development of probes designed for the report validation, ensuring the framework can effectively address real-world use cases and ensure it meets its objectives and delivers reliable results.

3.1 Vision

Keeping in mind the above discussion, there is a need for an approach in which the software system could be analyzed, undergo processing and produce useful information that can be used to provide enhanced insights about the software system. There

are several tools available for software analysis, each with its own strengths and weaknesses. However, the vision is to work on a framework which stands out due to some key differences. The most notable feature that we are looking for is platform and technology independence. Meaning the tool should not be tied to a specific technology. This flexibility will allow it to be written in any programming language, depending on the requirements, enabling data extraction from various software systems. Moreover, the tool should be able to run static code analysis on microservices and standalone services as well. Another distinctive feature is the integration of the extracted data with the unified data source. It should be capable of handling diverse types of data, generalizing it, and storing it in a graph-based database. This approach will eliminate the immediate need for a separate visualization to analyze the data visually. In cases where appropriate data representation is already available in the database, users can perform analyses directly without additional tools. Finally, our tool should not rely on a single visualizer for data representation. Since the data should be generalized, any compatible visualizer can be used with minimal adjustments to meet specific needs and requirements. This flexibility will enhance usability and ensures that the tool can adapt to diverse scenarios efficiently. In summary, the vision is that our tool should offers unmatched flexibility in data extraction, storage, and visualization, setting it apart from existing solutions in the field of software analysis.

In this report, we will discuss, implement and validate a framework by extracting static information from a project. In the future, this can be implemented/integrated with the CI/CD pipeline and provide dynamic information from the projects. The test project used in this report is Java spring framework based project called **petclinic**. Read more about this project from **spring petclinic microservices github**

repository (Spring-Team, 2025).

Moreover, our approach will be mainly focused on the unified data source (UDS) approach. It means that consistent, up-to-date and valid data will be available using the UDS technique.

> "Unified Data refers to the integration and consolidation of data from various sources into a single, cohesive framework. This approach allows organizations to streamline their data management processes, ensuring that all data is accessible and usable across different departments and applications. By unifying data, businesses can eliminate silos, reduce redundancy, and enhance the overall quality of their data analytics efforts". (Statistics-Easily, 2025)

3.2 Technical Strategy

This section provides a detailed discussion of the components in the framework. It also includes an analysis of the probes used to prove the framework's credibility, their use cases, and their benefits. After that, it discusses the unified data source (UDS) and, finally, the visualizer component. The framework has three main components.

- Probes
- Single Source of Truth
- Visualizer

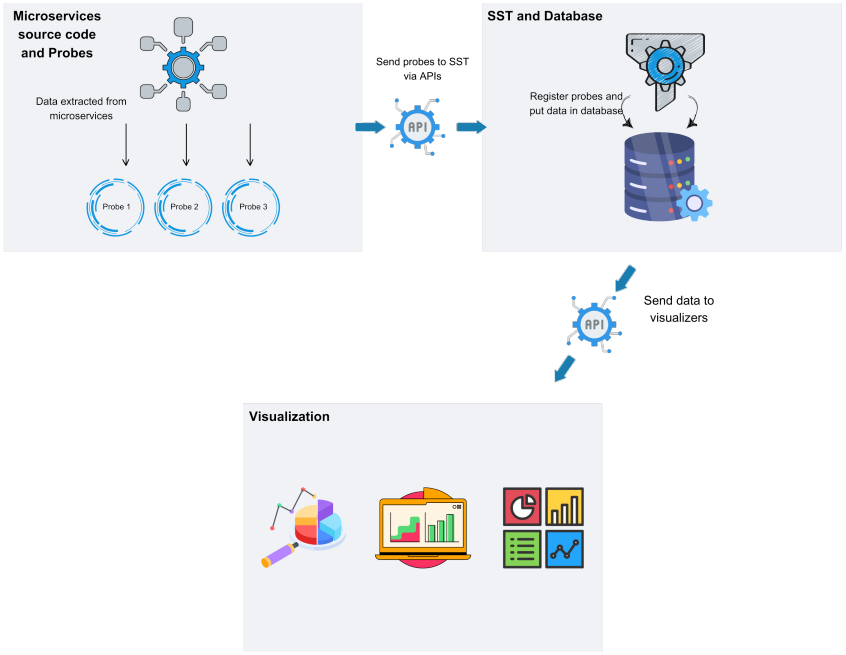

Figure 3.2: Working of Probes, SST and Visualizer

3.3 Probes: Extractors

The first component of the framework consists of **probes**. Probes represent distinct informational artifacts that are systematically extracted from software systems to provide insights and actionable data. In the context of this report, we have several use cases to extract specific pieces of information that are detailed in the below subsections. These probes will help prove the framework credibility and serve as the foundational elements for gathering critical data points that enable analysis and decision-making.

Looking forward, this concept can be expanded to more complex and targeted informational needs. By refining the scope and nature of the probes, we can tailor the probes according to our needs and to capture more refined data, addressing evolving

requirements and insights that drive meaningful outcomes. This flexibility ensures that as the systems grow or change, the framework remains relevant and capable of producing deeper, more impactful information.

Given below are some of the probes use-cases as part of this report in order to show the credibility of the presented framework. In later chapters, their implementation will be discussed at length followed by the validation of each scenario discussed here.

3.3.1 Authors Contributions

The Author Contribution Probe is designed to extract detailed information about the developers involved in modifying a specific method. It identifies the developer who made the most recent changes, provides a list of all contributors to the method, and highlights the developer with the highest number of contributions.

Use Cases:

- **Bug Fix**: When a bug is found in a method, the most recent author who updated the method would have the context of the recent changes that they have made and can help diagnose and resolve the issue faster. This will help the organization increase their productivity.
- **Reviews**: Looking at the most recent developer would give an idea of who to reach for documentation, clarification and review of the changes.
- **History Tracking**: Provides insights into who and why the person made the recent changes which can be used for documentation purposes.
- **Identify Developers Group**: Helps identify the group of developers who

worked on a particular method, which in turn enables them to share and transfer knowledge in case the top contributor of the method is unavailable.

- **Ownership**: Tracks if the code has been worked on collaboratively or primarily by one individual.
- **Refactoring**: Ensures that all the contributors can be consulted and their insights are taken before performing any major refactoring.
- **Knowledge and Accountability**: Identifies the subject matter expert for particular methods and assigns ownership of the method to the top contributor for accountability and guidance.
- **Planning**: Helps management decide who should be involved in any major changes to the method based on past contributions.

Benefits:

- **Improve Communication**: Facilitates and improve communication between team members by identifying relevant stakeholders.
- **Enhanced Planning**: Improves resource allocation, project planning and decision making for development tasks.
- **Transparency**: Promotes accountability among team members by making contribution history transparent.
- **Efficiency**: Increases team efficiency and issue resolving by involving right people for the job.

3.3.2 File Authors

This probe extracts information regarding the authors of a particular file. In software development, especially in large-scale microservices architecture, understanding the list of contributors to individual files provides significant value.

Use Cases:

- **Ownership**: Identifying contributors of a particular file highlights the ownership and familiarity with the functionality of the file of the people associated with it.
- **Collaboration**: Identifying contributors reveals whether a file has been developed collaboratively or by a single individual.
- **Auditing and Compliance**: Contributor information is crucial for tracking accountability and compliance. Specially in open-source projects.
- **Maintenance**: Files with multiple contributors might lead to maintenance challenges. Having list of contributors help assign the tasks to the correct people.
- **Bug and Issue Assignment**: Assigning issues to contributors who are familiar with the relevant files improves resolution speed.

Benefits:

- **Accountability and Quality**: Knowing who contributed to a file ensures accountability and encourages higher-quality contributions.

- **Team Collaboration**: Make team collaboration easier by identifying relevant stakeholders for discussions.
- **Reduces Risk**: Identifies files relying on single contributor which can help in workload distribution.

3.3.3 Author Relation

This probe extracts information regarding the strength of collaboration between two authors. Greater the collaboration, great will be the strength of bond between them. The "relation strength" metric quantifies the level of collaboration between pairs of authors based on shared contribution to files. In collaborative software development, especially in microservices architectures, understanding the strength of relation among authors can provide valuable insights into teamwork, communication and collaboration in the development process.

Use Cases:

- **Collaboration Analysis**: Provides a quantitative measure of collaboration between two developers in a development team. Higher strength shows frequent joint contributions.
- **Planning and Resource Allocation**: Helps plan tasks and form teams for future projects by using existing strong collaboration bonds.
- **Knowledge Transfer**: New developers can use relation strength data to identify key collaborators within the team.

- **Shared Ownership**: Helps determine the shared ownership of the files, making it easier to assign maintenance tasks.
- **Technical Debt**: Low pairwise strengths and high number of collaborators of a file represents cohesive ownership, leading to technical debt.

Benefits:

- **Improved team dynamics**: Improve team dynamics and encourages better interaction in areas with weaker team collaboration.
- **Fast Problem Solving**: Developers with high collaboration strength are likely to solve issues in shared files effectively.
- **Transparency**: Visualizes team interactions, increasing accountability and transparency.
- **Employee Evaluation**: Management can use the data to evaluate employees. Developers with higher collaboration strengths with multiple individuals shows employee value.

3.3.4 Microservices Endpoints

This probes extracts REST API endpoints from each service of a microservices project. Extracting such data provide valuable insights into how the application communicates and interacts with external systems or clients.

Use Cases:

- **Documentation and Analysis**: Automatically extracts endpoints information to generate accurate and up-to-date documentation and perform analysis.
- **Application Functionality**: Provides overview of the functionality exposed by each service.
- **Testing**: Extracted endpoint data can be used to perform regression testing and prioritize test cases.
- **Security Audits**: Endpoint extraction aids in auditing APIs for potential security vulnerabilities.
- **Productivity**: Helps developers understand the microservices quicks and API offerings of each service.

Benefits:

- **Debugging Issues**: Helps locating the faulty file and class, and helps developers quickly identify the method handling the request and resolve the issue.
- **Documentation**: Teams can use extracted endpoint data providing clients with API documentation.
- **Collaborations**: Facilitates communication between backend and frontend teams by providing endpoint insights.

3.3.5 Spring Beans

This probe extracts beans from java spring framework services. In Spring, the objects that form the backbone of your application and that are managed by the Spring IoC container are called beans. A bean is an object that is instantiated, assembled, and managed by a Spring IoC container. Otherwise, a bean is simply one of many objects in your application. Beans, and the dependencies among them, are reflected in the configuration metadata used by a container (Spring-Framework-Documentation, 2025).

Use Cases:

- **Debugging and Maintenance**: Bean extraction provides a clear map of all components, aiding in debugging and maintenance activities.
- **Debugging and Maintenance**: Extracted bean data reveals the dependencies between components, helping teams understand coupling within the application.
- **Dynamic Bean Management**: Helps to enable dynamic and conditional bean registration and configuration.

Benefits:

- **Improves Debugging**: Quickly locates missing or misconfigured beans, reducing the development downtime.
- **Security Improvement**: Detects potential security risks and stability issues.

3.3.6 Extracting Dependencies

This probe extracts dependencies from the POM (Project Object Model) files. A POM (Project Object Model) file is an XML file that serves as the fundamental building block of a Maven project. It contains information about the project and configuration details to manage dependencies, plugins, build settings, and more (Saurabh, 2023).

Use Cases:

- **Dependency Management and Version Control**: Extracting dependencies helps track the versions of libraries and frameworks used across microservices.
- **Microservices Dependencies**: Analyze dependencies to understand the relationships between microservices and their shared libraries.
- **Performance Optimization**: Shows performance heavy libraries or those that introduce inefficiencies.

Benefits:

- **Improves Performance**: Helps to identify, update or remove unused dependencies optimizing the performance and efficiency of the services.
- **Security Improvement**: Make it easy to identify those dependencies that are impacting the security of the services.
- **Maintainability**: Help keep all the dependencies up-to-date and easy to maintain.

3.4 Single Source of Truth (SST)

When data is being extracted from different sources, there are high chances of it becoming inconsistent and stale. If some sort of analysis is being done on the data, the input data needs to be up-to-date and reliable. In section 3.1 (Vision), we mentioned the visualizer component. It is supposed to run desired analysis on the data and provide visual updates. More on the visualizer in the upcoming subsection. So, if analysis is to be done on the input data, it needs to be a trusted, accessible, credible, reliable, and consistent.

Maintaining data quality is crucial for accurate insights and decision-making. Unified sources ensure that the input data remains clean and updated, regardless of its origin. Properly validated and processed data serves as the foundation for meaningful analysis and visualization.

(Müller et al., 2018) presents a scalable and extensible approach for software analysis and visualization by integrating data from diverse sources into a unified source. The paper discusses that having a unified data source provides a single access point for querying diverse data, eliminating the need to manage multiple disconnected sources.

The single source of truth (SST) component in the introduced framework does the same job as a unified data source. It is working as a data warehouse that takes all the information from different artifacts and process it. The tool used as a single source of truth in this project will be discussed in chapter 4 (Implementation Report) and the validation of the output provided will be discussed in chapter 5 (Scenario Validations).

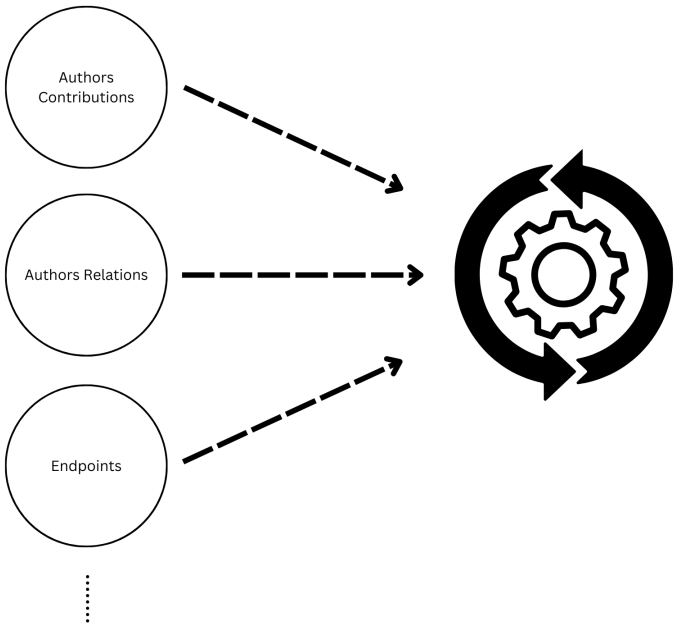

Figure 3.3: Probes and SST

3.4.1 Integration and Output

The probes will extract the required data from the microservices and feed them to the SST for further processing. SST will generate output which will be used by the visualizer for further analysis and visualization.

3.4.2 Data Management

The output of the SST will be stored in a separate database to maintain and preserve historical records systematically. This approach ensures that the data is structured, organized, and easily retrievable from a centralized repository. By consolidating the data into a single repository, it not only simplifies access but also ensures a consistent format, which is critical for long-term usability and reliability.

Storing the data in a database enhances its integrity by eliminating inconsistencies and reducing complexities, such as duplicate entries. This structure allows analysts or analysis tools to focus on extracting meaningful insights without the additional work of cleaning or reorganizing raw data.

Additionally, a dedicated database supports scalability, meaning it can accommodate larger datasets as the system evolves. It also offers improved data management capabilities, such as access controls and backup solutions, ensuring the data remains protected and readily available for future use.

3.5 Visualizer

After software analysis is completed with the help of probes and SST, we can use some visualizer tools in order to produce useful information gathered from the software system. There are bunch of tools that are already available provided by different software companies or we can create our own system in order to visualize the data as per need.

There are diverse range of software visualization and analysis tools available in the market, with each of them having their own strengths and weaknesses. (Bassil and Keller, 2001) covers more than 40 tools for software visualization with each having their own advantages. For example, Rigi is used for understanding legacy system, GraphViz is popular for graph-based visualizations and daVinci offers dynamic graph layouts.

Following are some of the important tools offered by AWS and Azure for visualization.

- **Amazon QuickSight**: A business intelligence (BI) service that enables users

to visualize data through dashboards, interactive graphs, and analytics. It connects to various data sources, including software systems, databases, and AWS services and perform real-time data analysis. Read more about this from (Amazon Web Services - QuickSight, 2025).

- **Amazon X-Ray**: A service for analyzing and debugging applications. It collects data about requests to your applications, including errors and performance bottlenecks. Visualizes the application architecture in a service map for easy identification of issues. Read more about this from (Amazon Web Services - X-Ray, 2025).

- **Azure Monitor**: Azure monitor collects, analyzes, and visualizes telemetry data from azure and on-premise environments to monitor application performance and detect issues. Key features are logs and metrics collection for in-depth analysis, provides alerts and insights to troubleshoot issues in real-time etc. Read more from (Microsoft, 2025).

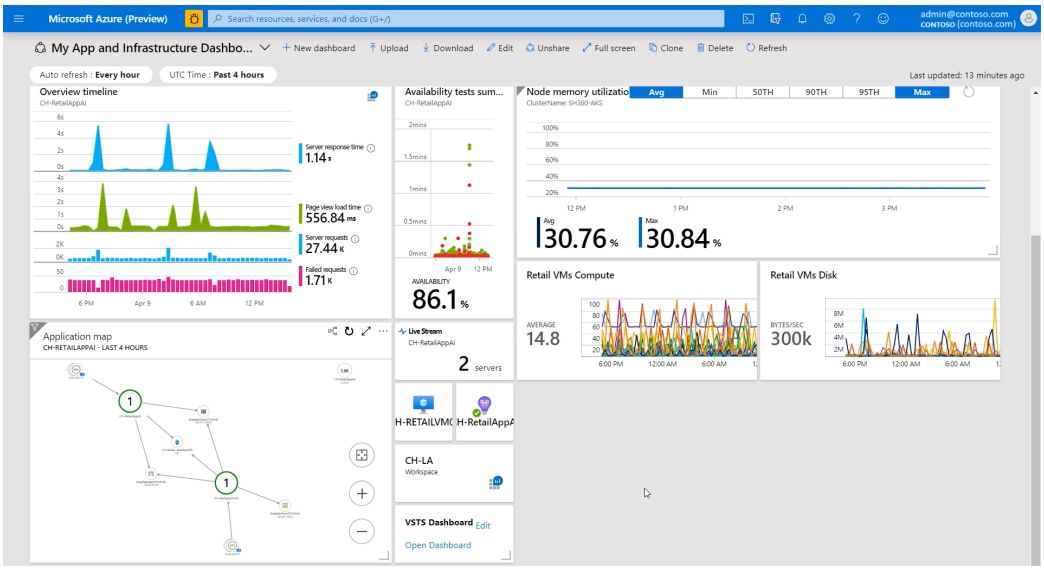

Figure 3.4: Dashboard of Azure monitor (adapted from Microsoft - Azure Monitor Best Practices (2025))

In our report, once the probes and SST are integrated, the processed data from SST will be used in for the visualization of data produced by the above mentioned probes.

3.6 Referencing

These are some sample references to GAMYGDALA (Popescu et al., 2014) from the `references.bib` file and state effects of cognition (Hudlicka, 2002) from the `references_another.bib` file. These references are not in the same .bib file.

3.7 Figures

This is a single image figure (Figure 3.5):

Figure 3.5: This is a single figure environment

This is a multi-image figure with a top (Figure 3.6a) and bottom (Figure 3.6b) aligned subfigures:

3.8 Tables

Here is a sample table (Table 3.1):

A	$\longleftrightarrow$	B
C	$\longleftrightarrow$	D

Table 3.1: A sample table

3.8.1 Long Tables

A sample long table is shown in Appendix B.

3.9 Equations

Here is a sample equation (Equation 3.8.1):

$$y = mx + b \tag{3.9.1}$$

(a) Figure 1

(b) Figure 2

Figure 3.6: A Multi-Figure Environment

Chapter 4

Implementation Report

Chapter 5

Scenario Validations

Chapter 6

Conclusion

Every thesis also needs a concluding chapter

Appendix A

Your Appendix

Your appendix goes here.

Appendix B

Long Tables

This appendix demonstrates the use of a long table that spans multiple pages.

Col A	**Col B**	**Col C**	**Col D**
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D

Continued on the next page

Continued from previous page

Col A	**Col B**	**Col C**	**Col D**
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D

Bibliography

Amazon Web Services - QuickSight. 2025. *Amazon QuickSight Documentation.* https://docs.aws.amazon.com/quicksight/latest/user/welcome.html https://docs.aws.amazon.com/quicksight/latest/user/welcome.html.

Amazon Web Services - X-Ray. 2025. *AWS X-Ray Developer Guide.* https://docs.aws.amazon.com/xray/latest/devguide/aws-xray.html https://docs.aws.amazon.com/xray/latest/devguide/aws-xray.html.

S. Bassil and R.K. Keller. 2001. Software visualization tools: survey and analysis. In *Proceedings 9th International Workshop on Program Comprehension. IWPC 2001.* 7–17. https://doi.org/10.1109/WPC.2001.921708

Gerardo Canfora and Aniello Cimitile. 2001. Software Maintenance. *Handbook of Software Engineering and Knowledge Engineering* 1 (01 2001). https://doi.org/10.1142/9789812389718_0005

Eelke Folmer, Jilles van Gurp, and Jan Bosch. 2005. Engineering Human Computer Interaction and Interactive Systems. In *Engineering Human Computer Interaction and Interactive Systems*, Rémi Bastide, Philippe Palanque, and Jörg Roth (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 38–58.

Eva Hudlicka. 2002. This time with feeling: Integrated model of trait and state effects on cognition and behavior. *Applied Artificial Intelligence* 16, 7-8 (2002), 611–641.

Microsoft. 2025. *Azure Monitor Documentation.* https://learn.microsoft.com/en-us/azure/azure-monitor/ https://learn.microsoft.com/en-us/azure/azure-monitor/.

Microsoft - Azure Monitor Best Practices. 2025. *Azure Monitor Best Practices - Analysis and Visualization.* https://learn.microsoft.com/en-us/azure/azure-monitor/best-practices-analysis https://learn.microsoft.com/en-us/azure/azure-monitor/best-practices-analysis.

Richard Müller, Dirk Mahler, Michael Hunger, Jens Nerche, and Markus Harrer. 2018. Towards an Open Source Stack to Create a Unified Data Source for Software Analysis and Visualization. In *2018 IEEE Working Conference on Software Visualization (VISSOFT)*. 107–111. https://doi.org/10.1109/VISSOFT.2018.00019

Adrian Popescu, Joost Broekens, and Maarten van Someren. 2014. GAMYGDALA: An emotion engine for games. *IEEE Transactions on Affective Computing* 5, 1 (2014), 32–44.

Saurabh. 2023. *Understanding the POM File in Spring: A Comprehensive Guide.* Dev.to. https://dev.to/saurabhnative/understanding-the-pom-file-in-spring-a-comprehensive-guide-4gnl#:~:text=A%20POM%20(Project%20Object%20Model,%2C%20build%20settings%2C%20and%20more. Accessed: 2025-01-13.

Spring-Framework-Documentation. 2025. *Introduction to the Spring IoC Container*

and Beans. Spring Framework Documentation Team. https://docs.spring.io/spring-framework/reference/core/beans/introduction.html Accessed: 2025-01-13.

Spring-Team. 2025. Spring Petclinic Microservices. https://github.com/spring-petclinic/spring-petclinic-microservices. Accessed: 2025-01-11.

Statistics-Easily. 2025. What is Unified Data? Understanding its Importance. https://statisticseasily.com/glossario/what-is-unified-data-understanding-its-importance/?utm_source=chatgpt.com#google_vignette Accessed: 2025-01-11.

Stratoflow. 2025. The Software Maintenance Process: What You Need to Know. https://stratoflow.com/software-maintenance-process/ Accessed: 2025-01-11.