BENEFITS SOFTWARE INTEGRATION HAS ON USER EXPERIENCE AND PRODUCTION EFFICIENCY

Lappeenranta–Lahti University of Technology LUT

LUT School of Engineering Sciences

Master’s thesis in Software Engineering

2025

Nestori Kangashaka

Examiners: Associate Professor Jussi Kasurinen &

Junior Researcher Katja Karhu

ABSTRACT

Lappeenranta–Lahti University of Technology LUT

LUT School of Engineering Sciences

Software Engineering

Nestori Kangashaka

Benefits software integration has on user experience and production efficiency

Master's thesis in Software Engineering

2025

44 pages, 3 figures and 1 appendices

Examiners: Associate Professor Jussi Kasurinen &

Junior Researcher Katja Karhu

Keywords: Software, Software integration, User experience, UX, production efficiency

This thesis investigates how software integration can enhance user experience (UX) and production efficiency in complex industrial systems. The research was carried out in collaboration with a company specializing in robotic welding automation stations, focusing on a software system developed to support the operation of these stations. The aim was to evaluate the extent to which integration could improve system usability and increase efficiency in production processes.

A literature review and a prototyping approach were employed. A prototype of an integrated system was designed and evaluated to assess its impact. The results show that integration increased system usability and automated tasks that had previously been performed manually. These improvements not only enhanced the user experience but also made the system more straightforward to use, thereby improving overall process efficiency.

TIIVISTELMÄ

Lappeenrannan–Lahden teknillinen yliopisto LUT

LUTin insinööritieteiden tiedekunta

Tietotekniikka

Nestori Kangashaka

Ohelmistointegraation hyödyt käyttäjäkokemukseen ja tuotannon tehokkuuteen

Tietotekniikan Pro gradu -tutkielma

2025

44 sivua, 3 kuvaa ja 1 liitettä

Tarkastajat: Apulaisprofessori Jussi Kasurinen &

Nuorempi tutkija Katja Karhu

Avainsanat: Ohjelmisto, Ohjelmisto integraatio, käyttökokemus, UX, tuotannon tehokkuus

Tämä opinnäytetyö tutkii, kuinka ohjelmistointegraatio voi parantaa käyttäjäkokemusta (UX) ja tuotannon tehokkuutta monimutkaisissa teollisissa järjestelmissä. Tutkimus toteutettiin yhteistyössä robottihitsausasemien automaatioon erikoistuneen yrityksen kanssa, ja se keskittyi ohjelmistojärjestelmään, joka on kehitetty helpottamaan näiden asemien ohjausta. Tavoitteena oli arvioida, missä määrin integraatio voi parantaa järjestelmän käytettävyyttä ja tehostaa tuotantoprosesseja.

Tutkimusmenetelminä käytettiin kirjallisuuskatsausta ja prototyypin tekemistä. Integroidusta järjestelmästä tehtiin prototyyppi ja sen vaikutuksia arvioitiin. Tulokset osoittavat, että integraatio lisäsi järjestelmän selkeyttä ja automatisoi tehtäviä, jotka aiemmin suoritettiin manuaalisesti. Nämä parannukset eivät ainoastaan vahvistaneet käyttäjäkokemusta, vaan tekivät myös järjestelmän käytöstä suoraviivaisempaa, mikä puolestaan lisäsi prosessien tehokkuutta.

ACKNOWLEDGEMENTS

I’d like to thank my professor for their guidance, patience, and encouragement throughout this project. My gratitude also goes to the team at the collaboration company for their support and insights during our work together. And finally, thank you to my friends and family.

SYMBOLS AND ABBREVIATIONS

UI	User interface
UX	User experience
gRPC	Google Remote Procedure Calls
API	Application Programming Interface
CI/CD	Continuous Integration/Continuous Deployment

Table of contents

Appendices

Appendix 1. Communication diagrams

1 Introduction

Robot automation has become a cornerstone of modern industrial production, offering a reliable way to increase efficiency, precision, and scalability [1]. Today, automation technologies are used across nearly all industrial sectors, significantly reducing the need for manual labour to complete repetitive tasks and allowing for more complex production to be undertaken [1]. To further enhance the effectiveness and adaptability of robotic systems, various software solutions have been developed and integrated into these automated environments.

These software tools are particularly important in production settings where robots must frequently adapt to changing tasks or product variations to operate correctly and efficiently. Manual reconfiguration in such cases is not only time-consuming but also increases the risk of human errors to occur. By automating these adjustments through software solutions that can either, do it automatically or by allowing for user inputs, when necessary, production lines can respond more quickly and precisely to change, while reducing the workload on human operators. [2]

As production demands grow more complex and dynamic, so do the software systems designed to support them. Often, multiple specialized software solutions are deployed within a single system that controls a single robot station, each having a specific responsibility to achieve precise action. However, for these systems to function effectively as a whole, seamless integration between the solutions is essential. Only when the solutions in the system are working efficiently together, can the full benefits from automation be harvested.

1.1 Background

This research is conducted in collaboration with Company A, which is currently undergoing a software integration process. The integration involves three distinct software solutions, referred to in this thesis as Software X, Software Y, and Software Z, which are together forming the system that is used to control a robotic welding station. While the specific functions of each software solution are beyond the scope of this research, it is important to understand the existing communication structure between them to grasp the need for the integration and see how the current challenges faced in today's software industry relate to the topic of this research.

Each software solution was originally developed to perform specific tasks independently. Especially when it comes to software X and Z. The software Y has always been developed as a solution that is used to control the robot, and it has the need to receive data from the other software solutions, even solutions that are not mentioned in this thesis. Over time, the communication between these solutions has increased and the potential value this specific integration would provide has become visible and, therefore, the need for tighter integration between these solutions has become increasingly evident.

The following diagram shows the current communication between the different software solutions. After that, how the data moves and what kind of roles each solution has will be explained. To fully understand the diagram, it is useful to know how the robot works. The robot can store data, so data can be sent to it and data, that it has created itself, can be read from it. The robot can be controlled by creating jobs that include instructions on how it should move and what kind of actions it should complete. The software Y is used to control the robot by telling it what jobs it should run and when. Software X and Z are used to create those jobs with both of them having different responsibilities in the job creation process.

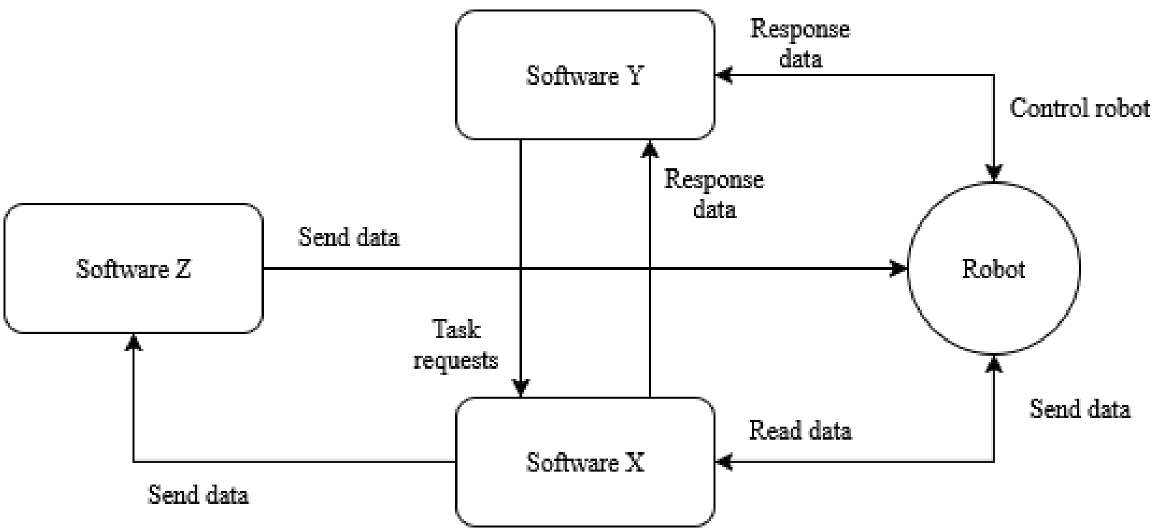

Diagram 1: Communication diagram of the current system

The main framework used for the communication channels between solutions in this system is gRPC (Google Remote Procedure Calls). Like can deducted from its' name, it is originally created by Google, and it is built on top of the RPC framework [3]. It is open source and can be used in multiple different types of applications, but in this system, it is used to create client/server HTTP connections [3]. The framework provides a lot of features that allow for the creation of secure and easily manageable connections that are easy to use, and future development will have a low effort. In gRPC the structure for the data that moves between the client and server is predefined, meaning that the client knows exactly what the response for a request, made to the server, is going to be [3]. This feature allows for the creation of simple and clear connections and situations where unexpected data was received cannot happen, increasing the security of the connections [3].

At present, Software X and Software Y have a direct communication channel that allows them to exchange data without manual user intervention. In this setup, Software Y acts as the controlling system, sending task requests to Software X, which processes the tasks and returns the necessary data. Before the software Y sends the task requests to the software X, it tells the robot to run a job that creates data required for the software Xs' tasks. When the software X receives a request, it reads the data from the robot and then completes the tasks based on the received request and sends the processed data back to the software Y. Usually, these tasks require for the software X to create jobs that are then also sent to the robot, so that the software Y can control the robot correctly.

As shown in the diagram above, software Z has no connection to software Y and vice versa. This lack of communication is a key issue in the current system, and one of the main goals of the integration is to address it. Currently the software Z can only receive data from the software X and send jobs directly to the robot. Because there is no automated communication between software Y and Z, the operator must manually monitor the current state of the station and move data when needed, adding extra tasks to their workload. This also has the potential to create gaps in the process which weakens the efficiency of the whole station and required extra error handling.

Currently, the system feels fragmented; while Software X and Y operate in sync, Software Z functions more like a detached component, despite being an integral part of the workstation. The goal of the integration is to seamlessly connect all three software systems, allowing data to flow between them while requiring minimal operator involvement. This would transform the current system from a collection of partially connected tools into a unified, interoperable system.

Operators manually use all three software solutions to complete various tasks. However, Software Y is used the most, as it controls the stations' movements. Many of the manual tasks performed with Software Z require user input and therefore cannot be fully automated. In contrast, while some tasks in Software X still require manual input, most can be automated.

A key motivation for integrating software Z closer into the system is to reduce or eliminate daily user interaction with Software X. By automating its core functions and enabling it to run in the background, the overall user experience would improve, and the workload on station operators would decrease. Although Software X would remain necessary for specific tasks, such as creating or modifying weld pattern data sets, its' usage would not be required on a daily basis, and the only users would be the welding engineers who have specific knowledge about welding.

1.2 Research Questions

This research aims to address the following questions, to what extent does UX improve with the software integration and how much does production efficiency increase as a result of the integration?

To answer these questions, the research begins with a comprehensive literature review on software integration, its benefits, and its impact on industrial automation. This review will provide a theoretical foundation for the integration process and identify the best practices and previous findings related to improvements in UX and efficiency.

Building on the insights from the literature review, a prototyping process will be executed to build a proof-of-concept model which will be examined within a controlled mock environment. This model will simulate the integration of multiple software systems in a robotic workstation and evaluate how well the more automated system performs and provide evidence for the predictions made during the literature review.

The findings from this study will inform recommendations on how the integration should be implemented in practice and help determine the level of resources that should be allocated to such an effort.

1.3 Limitations

The company for which this research has been conducted has requested to remain anonymous. As a result, certain aspects of the study must remain generalized. For instance, specific technical details regarding how the individual software solutions function will not be disclosed.

Additionally, the research is tailored to the specific needs and context of the company, which means that some perspectives and priorities reflect this particular industrial environment. Nonetheless, the subject of the research, software integration in industrial automation, is broad enough that the findings and insights can be applied to similar contexts, even across different industries, despite the absence of detailed explanations of the implementation.

1.4 Structure

Chapter 2 presents the literature review, which forms the theoretical foundation of the research. It starts with highlighting the usual challenges faced in software development in todays' software industry and after that explores existing studies related to software integration, UX, and automation in industrial environments.

Chapter 3 describes the prototyping phase, explaining what prototyping is and how it can be a helpful tool during software design processes. Also, the reasons for developing the prototype during this research will be explained, the used methodology will be shown, and how the prototype was implemented within the context of the research.

Chapter 4 brings together the insights from both the literature review, chapter 2, and the prototyping process, chapter 3. This chapter analyses the results and uses them to answer the research questions presented in the section 1.2 of this chapter.

Finally, Chapter 5 concludes the thesis by summarizing the key findings, reflecting on the research outcomes, and suggesting potential directions for future work. A list of references used throughout the thesis comes after the conclusion.

2 Literature Review

This chapter presents the literature review, which examines existing research related to the topic and evaluates how previous findings can inform and support the research questions outlined in Chapter 1. The review focuses on three key areas: software integration, UX, and the role of software in industrial environments. By analysing current knowledge in these topics, this chapter establishes a theoretical foundation for the study and identifies relevant insights that guide the development and evaluation of the proposed system. The topic of this research touches on the challenges that are currently faced in the software industry and for that reason, we will first go through them to get a better understanding of them and to clarify the context of this research.

2.1 Challenges in Software Development

Software development has become an increasingly complex and ever-changing field, facing a wide range of technical, organizational, and human related challenges. As software systems continue to expand in scale and complexity of usage, developers must address rising complexity, shorter delivery timelines, and evolving technological demands. These challenges affect every stage of the software life cycle and influence the quality, security, and sustainability of software systems [4].

Modern software systems are multifaceted, incorporating distributed architectures, cloud services, artificial intelligence (AI), and Internet-of-Things (IoT) technologies. This heterogeneity increases the difficulty of ensuring reliability, scalability, maintainability security and user-centred design [4]. The growing diversity of system components also heightens the need for robust configuration management and integration mechanisms to achieve robust products and efficient development. For small and medium-sized enterprises (SMEs), managing software variability and product lines remains a particularly demanding task, often constrained by limited resources, especially human, and formalized processes that are required for adapting to the variabilities [5].

The software industry continues to face pressure for accelerated delivery while maintaining cost efficiency. Agile and DevOps approaches have been widely adopted to meet these expectations, emphasizing iterative development and continuous delivery. However, these approaches can introduce challenges related to coordination, testing, and quality assurance, especially in large-scale projects where multiple teams must collaborate effectively [6], [7]. The tension between time-to-market pressures and long-term maintainability remains a critical issue in the industry and has been the cause for many failed projects but is not specific to the software industry.

Rapid technological innovation continuously reshapes practices used during software development processes. New paradigms, such as AI-assisted programming, low-code development platforms, and continuous integration pipelines, offer potential efficiency gains but also demand ongoing adaptation and learning from developers and managers [8], [9]. Constant technological shifts can lead to fragmentation in toolchains and knowledge gaps among team members, increasing the difficulty of maintaining consistent development standards. On top of this, usage of new technologies will make current solutions outdated, requiring continuous modernisation, further resources and robust designs.

Agile development has become a dominant methodology due to its flexible development processes and focus on customer collaboration. Nevertheless, applying agile principles effectively in large-scale environments remains a considerable challenge. Distributed teams often struggle with maintaining effective communication, synchronization of workflows, and shared understanding of requirements, all of which are important to achieve the wanted outcomes [6], [7], [8]. Managing dependencies, integrating deliverables across teams, and ensuring consistent quality are particularly problematic in such settings.

Software engineering has moved toward component-based and modular approaches for a long time, promoting reuse and flexibility [10]. While this standard supports scalability and maintainability, it introduces challenges in ensuring interoperability, managing dependencies, and maintaining version control across evolving, or legacy, components [4]. Effective component management practices are essential to avoid architectural erosion and to sustain long-term software quality.

Security and privacy have become critical concerns as software systems increasingly handle more and more sensitive data and operate in interconnected environments. Integrating robust security practices into agile development workflows is often hindered by competing priorities and time constraints [11]. Similarly, privacy management requires specialized expertise and active engagement from development teams. The introduction of roles such as “privacy champions” has shown promise in promoting privacy-aware design, but these roles face organizational and motivational challenges [12].

Many organizations still depend on legacy systems that are difficult to integrate with modern architectures and technologies. These systems often suffer from outdated technologies, limited, or also outdated, documentation, and a lack of scalability, posing significant barriers to modernization efforts [13]. Seamless integration and migration strategies are essential to ensure that modernization initiatives enhance performance and maintain operational stability without disrupting core business processes.

End-user expectations for software UX, performance, and personalization have increased substantially in recent years. Delivering high-quality UX requires thorough collaboration between the managers, design, development, and testing teams. AI-driven development tools are beginning to support this process, but their integration into workflows raises issues of transparency, reliability, and trust [9], [14]. Managing user expectations while balancing functional and non-functional requirements remains a persistent challenge in software engineering. But, as with security, matching user expectations is one of the first aspects to lose resources when fast development is required.

Next, we will be discussing what software integration is and how it is beneficial for modern software systems.

2.2 Software integration

Software integration is a process where two or more software solutions are integrated into a single system where the software solutions share data and work together to solve problems [7], [15]. The complexity of software systems is increasing all the time and to keep the development of these systems manageable, the systems must be separated into individual software solutions where each solution has its own responsibilities. To form the final system, these separated solutions are then integrated together to fulfil the use cases set for the system.

With the continuous increase in the complexity of software systems comes increasing developmental complexities. To combat this, some developers have started to create ready-to-use modular software components that can be integrated into other larger software systems. These components can be shared as open-source software components, or they can be sold for the developers that want to reduce their own development costs and integrate ready-to-use components into their system. The main benefits of integrating software components acquired from external vendors or sourced from open-source libraries include cost savings, reduced time-to-market, access to multi-skilled workforce, innovative best-practices and proximity to market and customers. [4]

On top of integrating separate software components into a system, software integration can be done to increase the communication between two or more separate software solutions [16]. These solutions can be developed by the same developers, or the integrated solution can be an open-source software or a bought solutions from a vendor [16]. This research is conducted to support a situation where the integration is happening between solutions that are all developed by the same developers, so we will be only focusing on that kind of situations.

Increasing the communication between software solutions by integrating them into a system where responsibilities and use cases are divided clearly and development is done separately; the structure can be kept simple while the complexity of the system and its' use cases increase [5], [17]. The need for the integration comes from the need to fulfil new use cases or the need for the current use cases to be handled more efficiently is required. This can mean that the communication is increased between the solutions in the current system or that new solutions are introduced into the system so that those new use cases can be satisfied. Important thing to note is that the ability to fulfil the old use cases should not be removed from the integrated system. The only situation where old use cases can be forgotten is if the new use cases make them redundant.

Maintainability is one of the most important aspects of modern software systems and their development, because agile developmental practices require for the continuous implementation on top of old developments. By separating the responsibilities of the software system into separate solutions and developing them separately, the maintainability of the system can be increased [16]. This reduces the cost of maintenance in the future and the resources required by adding new features into the system [16]. On top of this, it allows for smoother modernisation of the system in the future, which is important in the current state of the software industry.

To follow this, we will discuss the effects of UX on software systems and the opportunities it offers for software development.

2.3 User Experience

It is hard to give a definitive definition for what UX means [18]. Some definitions state that UX is only about the subjective feelings that the users experience during the usage of a system and objective measures won't give accurate views on the quality of the UX [18]. Some state that UX also includes the efficiency and effectiveness of the system as key aspects of good UX [18]. However, we will define UX as follows; good UX means that the software system in question is intuitive to use and easy to learn [9], [18]. The ideal situation is that the user doesn't need long instructions or time-consuming tutelage, but the usage of the system is so intuitive that it's more complicated to do it wrong than right.

When a software system is used to enhance industrial production, where there are other stepping stones, like the complexity of the process the software is integrated into, adding new ones with the systems internal complexity is completely pointless. Especially when some of the operators of those systems have been used to using certain tools for decades and has now been taught new tools and are forced to forget the old ones. In some cases, the software has to be of some level of complexity, because the complexity of its' use cases requires it to be, but it should always remove complexity from the use cases it is applied to [18]. To increase the usability of a software system that is integrated into a complex process, all the terms and wordings should be shared with the ones that have been used in that sector of the industry. This removes the need for the new users to learn a new vocabulary, and they immediately know what the words mean. This methodology should also be applied to colours and iconography, if they are present in the context of the software system.

UX is a widely researched field and is one of the most important parts of software systems, but it keeps getting overshadowed during the planning of systems by other aspects that seems to be more important [19]. The lack of interest and focus on UX during the planning process means that it won't get the required amount of resources allocated towards it and it won't have as high of a level as it should have. Lack of attention on UX directly correlates with decrease in maintainability and the overall quality of the system.

When a software is created using agile and continuous integration and development (CI/CD) tools, like most software systems are today, maintainability becomes one of the most important non-functional aspects of the whole project. CI/CD development is built on the concept that new features are continuously added on top of the old ones [20]. When the system has a lot of code that needs to be deciphered to add new features, the development process will become increasingly more complex and the chances for new bugs appearing increases [21]. To match the requirements for fast development, these issues are ignored, and new code is added on top. Bad code created on top of bad code overtime creates issues that are so hard to fix that creating a new system becomes a better option over fixing the current solution [21]. To ensure that that doesn't happen, the new features must be of high quality, and the overall quality of the whole system must be kept up to date at all times [21]. This can be achieved by incorporating good testing and quality assurance tools and processes into the development process. This makes it possible for the developers to keep adding innovative features into the system that increases the marketability of the whole solution.

Paying attention to the UX of the system and improving it makes the quality of it higher [18]. This creates a marketability that is important when the system is used to make complex processes more efficient and easier to handle [22]. This is the case for software systems that are integrated into the industrial processes, like robot automation. Making the process of controlling robots with software systems easy, increases the efficiency of the process and reduces wasted resources, which in turn increases the user satisfaction and the recognition from potential clients. If a potential client can get a hang of how the system is used during a short demo, there is a lot higher chance that they become clients of the product and are interested in other solutions that the company is offering [22].

When a system includes the usage of multiple separate software solutions, the unity of their User Interfaces (UI) becomes important. When all the different software solutions share the same UI design philosophy, the UX of the system increases [23]. Having the same style in all applications makes learning the usage of the solutions a lot easier and makes the transition from switching between the solutions a lot smoother [23]. Even when the users must switch between software solutions, granting them the feeling that they are still using the same system reinforces the quality of the whole system and makes it easier for the users to understand what certain aspects of the software is for.

Conversely, when this unity is lacking, an inconsistent UI, where visual or interaction patterns vary across components, for example buttons and text fields, risks undermining user confidence and perceived product polish. Research indicates that consistency enhances learnability and usability, increasing satisfaction even when the functionality stays the same [17], [24]. Additionally, large-scale development teams address UI inconsistency via design systems and style guides to reinforce cohesion and reuse [24]. From a psychological standpoint, inconsistent interfaces elevate cognitive load, reduce trust, and may appear unprofessional or demo-like, traits incompatible with marketable, innovative systems [24].

The final section of the literature review discusses how software systems have been implemented into industrial environments and how they can improve the levels of automation and efficiency.

2.4 Software in Industrial Environments.

Industrial environments have benefited immensely from the digital revolution and the improved automation it has the ability to provide [25], [26]. The integration of software systems into industrial processes has transformed operations and expanded the possibilities that lie within industrial automation [26]. These systems not only increase productivity and quality but also enable the handling of more complex problems than before, which makes it possible for industrial problem solving to increase the quality of environmental causes [25]. To meet the growing demand for innovative solutions, the importance of efficiency and quality in industrial processes cannot be overstated.

Environmental and regulatory requirements have driven the need for efficient, low-waste production. This is increasingly achievable through software-based industrial automation, whose capabilities increase with the advances in information technology on top of the advancements from the industrial industry. By streamlining repetitive tasks and reducing human intervention, automation decreases the likelihood of human error, thereby improving production quality and minimizing wasted resources. Furthermore, integrating data-driven software solutions allows for continuous monitoring and analysis of production processes granting more information about the processes [27]. The resulting data can be used to support informed decision-making, optimize workflows, and further reduce material waste caused by wasteful production and avoidable errors [27].

Automated systems follow predefined instructions with high accuracy and consistency, unachievable by humans, reducing the number of defects and enhancing system reliability. This predictability makes it possible to forecast production levels for specific time periods, enabling more effective planning and coordination with processes that are directly in correlation with it [27]. Continuous and consistent operation further strengthens production predictability, supporting the streamlining of scheduling and supply chain activities.

In welding automation, achieving consistency is especially critical [28]. A central challenge lies in the variability of the workpieces that are being welded on. Even minor deviations in them require recalibration and can compromise weld quality [28]. Automated welding systems must therefore be capable of sensing, analysing, and adapting to individual workpieces in real time to achieve weld quality that is satisfactory [28]. If the system cannot ensure sufficient adaptability, higher productivity and efficiency loses its value, and manual welding may still be preferable, even when the cost and time it requires is higher.

Nevertheless, the potential benefits gained from automated welding are substantial. Manual welding is labour-intensive, physically demanding, and time-consuming, with higher error risks due to fatigue [29]. Automation not only offloads these burdens but also allows human operators to focus on supervising and optimizing the process rather than performing the welds themselves [29]. Additionally, the global shortage of skilled welders increases the necessity of automation, not only as a means to boost productivity and efficiency, but also as a practical requirement for sustaining production capacity [29].

In summary, the literature review highlights the challenges and opportunities that lie in modern software development and its’ processes and standards. As software systems grow more complex, decentralisation of software solutions grow more important, as they allow for more modular designs and architectures that are of high importance for agile development, and therefore efficient integration between those solutions becomes important. To achieve reliable and efficient software systems, effective integration practices and standards, strong focus on UX and user-centric design, and careful consideration of maintainability becomes essential. Modern software systems handle a considerable amount of private data that requires for the security of the systems to be a top consideration and the transparency regarding what data will be stored, why it is stored and how it is going to be used. In industrial environments, software plays a large role in advancing automation. It increases the possibilities withing modern industrial manufacturing, makes the workflows more efficient and enhances the productivity of the production. On top of this, it helps with gathering data about the production, which can be used to enhance the production of the entire manufacturing chain and helps with finding and preventing possible points of errors. The insights gained from the literature review form the theoretical foundation for this research and directly informs the design and evaluation processes of the proposed prototype that will be discussed in the following chapter.

3 Prototyping

Following the completion of the literature review, a prototype of the designed system was developed to test and evaluate it in practice and to provide further proof for the predictions made during the literature review. This chapter explores the concept of prototyping in greater detail, including its purpose within the research process, the methods used, and the rationale behind selecting prototyping as a research approach. It outlines how the prototype was implemented and how it fits into the overall research design. The outcomes and insights gained from the prototyping phase will be analysed in the Chapter 4, alongside answering the research questions and the broader findings of the study.

3.1 What is it?

Prototyping is an educational and useful step in the software design process and plays a key role in developing high-quality software systems. It is a widely adopted practice used to evaluate new design concepts, test proposed solutions and communicate ideas for new features or architectural changes within existing systems [30]. By creating early models, teams can explore new ideas and experiment with different types of approaches before fully committing resources, helping clarify requirements and align stakeholders on shared goals [30]. It is a collaborative process that doesn't only increase innovation but also enhances the chances of delivering software that actually meets the requirements set by the stakeholders.

At its core, prototyping involves creating a tangible or functional representation of a theoretical design [30]. Instead of relying solely on design documentation or verbal descriptions, teams build interactive models or simulations that show the key features and workflows of the system in question. This hands-on approach enables designers and stakeholders to visualize and interact with the system, revealing how it behaves in practice from multiple perspectives [30]. This engagement with the designers and stakeholders helps with identifying strengths and weaknesses, uncovering usability challenges, and exposing potential integration issues early in the development process, guiding more informative and efficient design decision making [30].

Prototypes can vary significantly in both scope and fidelity depending on the goals and constraints set for the prototyping process [30], [31]. They may represent an entire system or focus on specific components or subsystems [31]. Particularly in early design phases, where targeted evaluation of critical interactions is more efficient, this is a widely used way to get more out of the prototyping process [31]. Narrowing the focus prevents teams from being overwhelmed by complexity and allows them to prioritize resources on areas with more impact. Similarly, the level of detail can range from quick, low-fidelity sketches or wireframes designed to gather initial feedback and high-level ideas, to high-fidelity, interactive prototypes that closely mimic the final product and may even evolve into the working software [30], [31]. Selecting the appropriate scope and fidelity ensures that prototyping efforts align with resources allocated for the project and the complexity of the system and its' use cases.

Within the context of this research, prototyping serves two purposes. Validating the theoretical design of the software integration and providing a concrete environment in which user experience and process efficiency can be observed and assessed. By interacting with a prototype, stakeholders gain valuable insights into real-world workflows, potential bottlenecks, and usability improvements [30]. These are insights that are often overlooked when only studying theoretical models. This practical evaluation is essential for refining both the technical and human factors critical to successful software integration implementation [30].

The level of detail, functionality, and resources dedicated to a prototype are guided by the specific needs of the project. Factors such as time constraints, technical complexity of the system and therefore the prototype, and available development tools all influence how the prototype is constructed and how closely it resembles the final system [30]. Balancing these considerations helps maximize the prototype's value as both a learning tool and a communication medium without over-developing the prototype and overextending project resources.

Importantly, by prototyping early, development teams can reduce the risk of costly redesigns later in the process [30]. Early feedback and insights often reveal design flaws or user concerns that, if left undiscovered until later stages, would require significant rework. This approach enables iterative refinement, streamlines development, and ultimately leads to a higher-quality product delivered more efficiently and with fewer surprises.

Now that the concept of prototyping is clear, the reasons for using it as a development and research tool will be discussed.

3.2 Why it’s important?

Prototyping serves as a critical testing method within the design process of a software system or component, making it an extremely valuable step in developing quality software [30]. By thoroughly executing prototyping, many potential design issues can be identified early, before the development phase begins, thereby facilitating a smoother and more efficient redesign and development processes [30]. This early validation also increases designers’ confidence, as they gain assurance that their designs are correct and the confidence of the developers as they also have confirmation that the design is correct and the only thing that is left to do is the development of the design.

Uncertainty regarding client requirements is a common challenge in software projects. Often, clients may struggle to precisely articulate their needs or envision the final product. This might be caused by the lack of software technical knowledge from the stakeholders’ part. In such cases, prototypes provide valuable clarification, enabling clients to better understand and communicate their expectations [30]. This iterative feedback loop between clients and developers helps ensure that the delivered product aligns closely with the client’s true requirements and needs [30].

Effective communication between clients, developers, and other stakeholders is another vital aspect of successful software design and development. Prototyping significantly enhances this communication by providing a concrete reference points for discussions. Strong, transparent communication built around prototypes helps align expectations and fosters confidence that the final product will meet the intended goals [30]. For some stakeholders, prototyping can seem like a waste of time and resources, but a good communication can help with making it easier for them to see the full benefits of the process.

All of these benefits collectively reduce the risks inherent in software development projects. Prototyping encourages experimentation, which aids in uncovering technical challenges, design flaws, and unfeasible ideas early on [30]. By revealing such issues ahead of full-scale development, prototyping helps save time and resources that might otherwise be spent on costly rework or failed implementations.

Next the reasons for selecting prototyping as a research method for this research will be explained.

3.3 Goals

Prototypes are typically created during the design phase of new software solutions, but they are also essential when significant modifications are made to the architecture of an existing software solution or to the architecture of a system that includes multiple software solutions [30]. Such changes often require a thorough inspection and validation before the new components or integrations are fully developed. Given that this research focuses on a software integration process, prototyping was employed as a key method to support this exploration.

For this research, the prototyping had two goals. The first one being to assist in answering the research questions by providing a practical representation of the integrated software system's communication within a controlled, mocked environment. And the second, giving the stakeholders a better understanding for how the proposed system would work. With the information gathered from the prototype, the stakeholders get confirmation about the concept that they had for the integration and a more informed decision can be made on how the integration will be developed.

The prototype was not intended to be an exact replica of the final designed solution but rather to simulate how data and commands would flow between the integrated solutions. This approach allowed for focused evaluation of the integration's core functionalities, user interactions, and potential bottlenecks without the complexity of a full-scale implementation, relieving resources and allowing for the process to be done quicker.

Furthermore, the prototype was developed with scalability in mind, designed to evolve into the final system should the design be approved by the stakeholders. This required building the prototype on a foundation that supports future expansion and refinement. This approach was taken, because the prototype is made for a system that already includes full-fetched solutions and the creation of the prototype required for the implementation of actual features that the proposed system would include and building on top of them makes the potential development phase more efficient, while not adding into the workload of the creation of the prototype.

By aligning the prototype's design with the eventual development roadmap, the process aims to reduce the transition time from prototype to production-ready system and to ensure continuity in addressing the identified requirements. This also means that the issues found in the prototype are issues that would be directly related to the final product, making it a more informative for the current context.

Now that prototyping as a process is understood and the reasons for it to be used a research method for this research is clear, we will discuss how the prototype was implemented.

3.4 Implementation

To get a better understanding on how the prototype can be used to answer the research questions, lets' go over how the implementation of the prototype was conducted in the context of this research. This gives valuable insight into chapter 4 where the findings from the prototyping is explained with the findings of the literature review.

To create the prototype, full features were added into the software X and Z, and the software Y was completely mocked, meaning that the real software was not included into the prototype. This is because the prototype is inspected from the point of view of the software Z and the work for the prototyping process decreased when the Y was left out and including it wouldn't increase the information gathered from the prototype, allowing for the process to be concluded in a faster schedule.

Full features were added into the software X because its' usage after the integration decreases and this was important to present in the prototype. With the full implementation of the software X, the future usage, which will be occasional and not done by the operators themselves but rather by welding engineers, was highlighted more clearly. If the integration is going to be developed, the full implementation of the software Xs' features during this phase means that the core structure for the software is already done, and more attention can be given to the other software solutions that have a larger development roadmap. Implementing full features means creating a new Application Programming Interface (API) for the Software X that the software Z can use to send requests and receive data. This was done by implementing a gRPC- connection between the two solutions in a way that the software X acts as the server and the software Z as the client. The reason for using gRPC- connections is that they are already used in the system, for example, for the connection between software X and Y. Some minor changes might be required to add into the software X, but other than that it has everything that is needed for the integration to be deployed.

Three features were added into the software Z. Client sides for gRPC connections to the software X and Y and then internal logic where data received from the software X is used and then sent to the software Y. The detailed functionality of the internal logic is not important and won't be discussed in this research.

Like was said before, the software Y was completely mocked, meaning that the real software Y was not included in the prototype. However, a simplified version of that software was used, that had been created prior to this research, where the gRPC- connections between the software X and Z were completed. The connection to the software Z was created completely for this prototype, but the gRPC- connection to the software X was already in use. The integration only required modifications into that connection and not a creation of a new one. This means that the mocked software had the ability to receive data from the software Z and then send and receive data from the software X. This allowed for the simulation of the complete communication triangle between the three software solutions. Below is a diagram of the communication channels in the prototype. It is built on top of the diagram 1, that was shown in the background section of the introduction chapter.

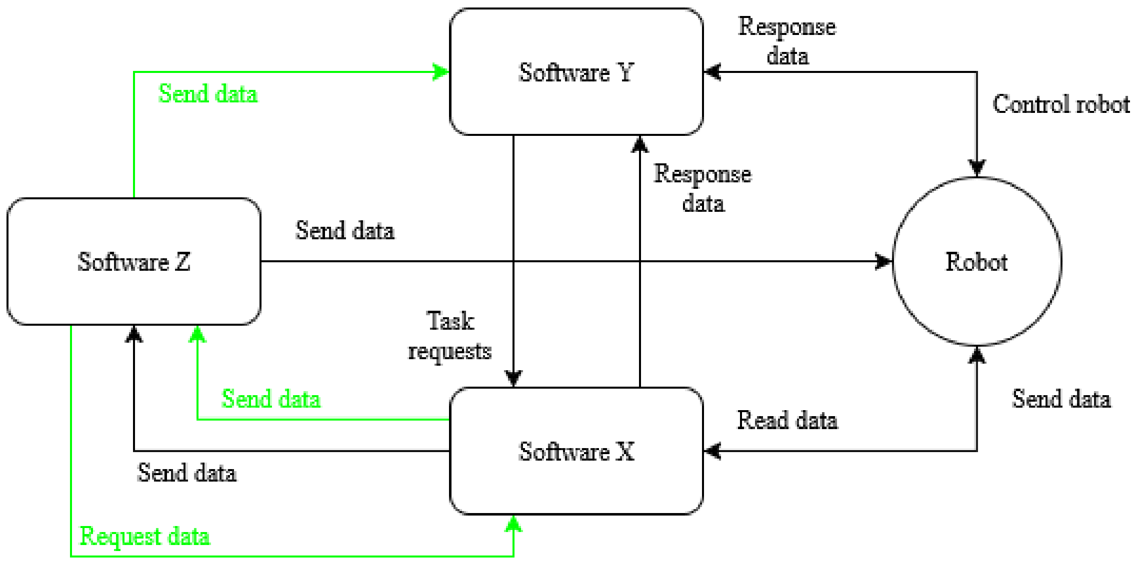

Diagram 2: Communication diagram of the prototype

New connections, that were created for the prototype, are highlighted with green on the diagram. As can be seen, the communication now forms a full circle between all software systems included and the robot. The black connection between software X and Z is left there for backwards compatibility reasons and new gRPC connections were added, instead of improving the old non-gRPC- connection, to make the system more unified in terms of the technologies used. The old connection moves data that does not relate to the data moved through the new gRPC- connection and therefore was not modified during this development. The technological unity, achieved from incorporating gRPC- connections with all the other software solutions, allows for the systems' future maintainability to require less effort and research. Even thought, new changes were added into the connections between software X and software Y, they are not highlighted in green because they are not new connections and therefore are not new in terms of the whole communication of the system and neither in the technologies that are used in them, as they have already been made using gRPC.

In summary, the prototype was made by adding features into the existing system. Full features were added into the software X and Z, but the software Y was completely mocked, meaning that the real software was not included into the prototype. The prototyping process provided practical insight that could be used to validate the predictions made during the literature review and the theoretical design for the integrated system. Through the implementation process of the prototype, valuable insights were gained regarding improvements towards the system’s UX and production efficiency. The outcomes of this phase form the basis for the analysis presented in the next chapter, where the findings derived from both the prototype and literature review are discussed in relation to the research questions.

4 Research Findings

This chapter presents the key findings of the research. It draws on the information discussed in Chapters 2 and 3 to address the main research questions that are; to what extent does UX improve with the software integration and how much does production efficiency increase as a result of the integration? It also ties the topic of the research into the opportunities and challenges faced by the software developers in today's industry.

4.1 Overview

This research is closely related to the challenges faced in today's software development industry**.** As discussed earlier in the literature review, modern software engineering faces increasing complexity, higher security needs, tighter delivery schedules and rising expectations for quality, efficiency and maintainability [4], [9], [11], [14]. These same challenges are similarly visible in the industrial automation industry, where software systems must interact seamlessly with physical machinery, which brings its' own complexity and regulations to the table, different kind of control systems, and various external interfaces that can differ technologically by a large margin from each other. The need for reliable integration, efficient and secure data exchange, consistent and effective system performance and high level of usability is exactly in the middle of the software industry wide challenge of achieving fast development times with high-quality levels and long product lifecycles.

In industrial automation environments, these challenges become similarly evident by strict operational requirements and legacy dependencies, that are intensified especially during integration processes between modern and legacy systems. Many automation systems rely on legacy components or sub-systems that are built using older technologies that increase the complexity and required effort of the integration processes. This situation parallels the general software engineering issues of maintaining interoperability and managing component-based architectures, as discussed during the literature review [13]. Furthermore, the demand for high efficiency and product quality, both in manufacturing processes and the supporting software systems, requires that the integrated solutions must be robust, maintainable, secure and have user-centred design, both, as stand-alone solutions and as a unit where everything works together.

UX expectations also play a significant role in the challenges faced by the software industry today. Operators interacting with automation systems depend on intuitive and consistent software interfaces, so that the systems are not affecting the efficiency of the processes negatively. Addressing these expectations links directly to the broader challenge of meeting UX and reliability requirements in software development. In this research, these considerations shaped the design and evaluation of the prototype of the integrated system, ensuring that the developed solution not only supports functional integration but also enhances overall UX and workflow efficiency. Without the added benefits to the UX and workflow efficiency, the benefits from the integration might be enough for it to be developed.

By examining software integration within an industrial automation setting, this research contributes practical insight into how these software development challenges manifest in a domain where UX and system quality, security and interoperability are in a central role. Therefore, the findings provide both theoretical alignment with existing software engineering literature and actionable evidence for organizations facing similar integration challenges and plans.

This research was conducted for a company undergoing a software integration process. The purpose for the research was to provide actionable evidence about software integration and to answer the research questions set by the company so that they can make an educated decision about how the development of the designed integration should be conducted and what kind of possibilities lie within the integration process. This information is crucial for making the right decisions about how the integration should be developed, what kind of priorities different aspects have and how much resources are needed by the development process. On top of that, further understanding of the benefits an integration of this type and scale could have, will be extremely helpful for the future development of the whole system for the company, and possibly for other integrations between other software systems that the company offers.

The main research methods used to address the research questions were literature review and prototyping. A literature review was conducted to get a better understanding about what have been studied about the subject, how that knowledge relates to the current topic and if there are some aspects of the topic that require further researching. After the literature review was conducted, a prototype was made that simulated the integrated system in a mocked environment. Reasons for using prototyping as a research method were to achieve concrete and demonstrable evidence about the integration and to give the stakeholders a more thorough idea of the proposed future for the system, since literature review alone cannot provide that. Prototyping allows for the studying of the expectations gained from the literature review. On top of this, the prototyping allowed for a smoother transition into development since there were already functionalities in the prototype that are included in the final product.

First the literature-based evidence will be presented. After that the concrete evidence from the prototype will be discussed. And finally, the research questions will be answered based on those findings.

4.2 Literature Review Overview

When the problem, that a software system is created to address, reaches certain level of complexity, the structure of the system needs to be decentralized for the development and usage of the system to stay simple [7], [15]. This means that the system will include multiple separate software solutions that are all created to address single responsibilities and are then integrated together to form the complete system. This allows for the development of the system to stay simple, because each solution is developed separately and are only connected to each other through APIs.

In some cases, integration happens between existing systems. This means that, either an existing system, that has previously been completely separated, is now integrated into the system, or that the integration is needed to increase the communication between the solutions that are already part of the same system [16]. Since this research focuses on software integration and its benefits in a situation where communication between existing solutions is enhanced, the following overview will concentrate exclusively on the literature from this context.

Even though software solutions remove the need for human interactions for the most parts, some are still required, especially when the system is used to address a problem that is complex. This is the case with the software system that is in the middle of this research. When the system requires user inputs and is decentralized into multiple sub-systems, the usage of it might suffer, because the user input is required in multiple solutions. This creates a situation where the user has to switch between the different solutions, which in turn increases the risk of breaking the context of the system and allows for the system to feel discarded.

The unity of the UI between the solutions is important because switching between applications that have different UI philosophies is going to break the immersion that they are all part of the same system [17], [23], [24]. This decreases the usability of the system and makes it harder for the users to learn the usage of it. Especially when the software is used in a context where certain standards for the way things are worded and specific meanings for symbols exists. Differing styles also decrease the trust that the user has on the system, and it might cause potential clients to switch to another provider that offer a more unified solution [17], [23], [24].

If the redesign and redevelopment of the solutions in the system to match each other, UI philosophy wise, is not feasible, the issues coming from differing UI philosophies can be alleviated. By increasing the communication between the software solutions, the need for user input can be shifted between the solutions, removing the need for the users to switch between the applications [5], [17]. This might not remove the need for the user input, but it will remove the need to switch between applications which will increase the usability of the whole system. It is important to note that this does not remove the issue, that is the difference between the UI philosophies, but it decreased the points where it is realized.

Several themes emerged during the literature review that directly inform this research. Previous studies emphasized that modular and integrated architectures improve maintainability, scalability, security and communication efficiency within complex systems [4], [5], [10], [16]. However, they also highlighted that integrating multiple software solutions together introduces new usability challenges, particularly when separate software solutions lack visual or functional consistencies [17], [23], [24]. Prior research on software systems in industrial settings further established that UX of software systems have a measurable impact on error reduction and production efficiency in automated environments [18], [22], [25].

Despite these insights gathered from the literature, limited empirical evidence, on how integration affects UX and production efficiency when all subsystems are developed by the same organization, was revealed. Most prior studies focused on cross-vendor integrations or technical performance rather than the point of view of the end-user. This gap shaped the direction of the present research: to evaluate, through a prototype, whether increased communication between in-house solutions can both improve UX and production efficiency.

Therefore, the findings presented in this chapter not only build upon the theoretical expectations identified in the literature but also aim to test their validity in a practical industrial context. The prototype developed during this research works as a way to materialize the findings of the literature review and provide concrete evidence if the integration of the systems does increase the UX and production efficiency in practice and not just theory.

4.3 Prototype Overview

To represent the proposed system after the integration, a prototyping process was used. During that process, a prototype of the proposed system was created. The prototype simulated the system after the integration in a mocked environment, and the goal was to gather information that could be used to answer the research questions and to give the stakeholders a better understanding of what has been proposed so that an educated decision can be made about the future of the system.

The prototype was created in a way that it can evolve into the designed system. This was done to decrease the time and resources required by the development phase after the prototyping phase. Since the prototype was conducted for a system that already exists, there already were features and functionalities that could be used to create the prototype and it was the easiest and most useful way to conduct the process. This also helps with making it easy for the stakeholders to see what the system would actually look like, since the basis for the prototype was the system before the integration.

The whole prototype includes three different software solutions and full features of the designed system were added for two of them. One of the software solutions was completely mocked, meaning that the real software was not included into the prototype and just a simplified version was used. The goal for the prototype was to simulate the improved communication between the software solutions that the integration would provide and by implementing the full features for the two software solutions this was possible to present.

The following diagram shows what kind of changes the integration brings into the communication of the system.

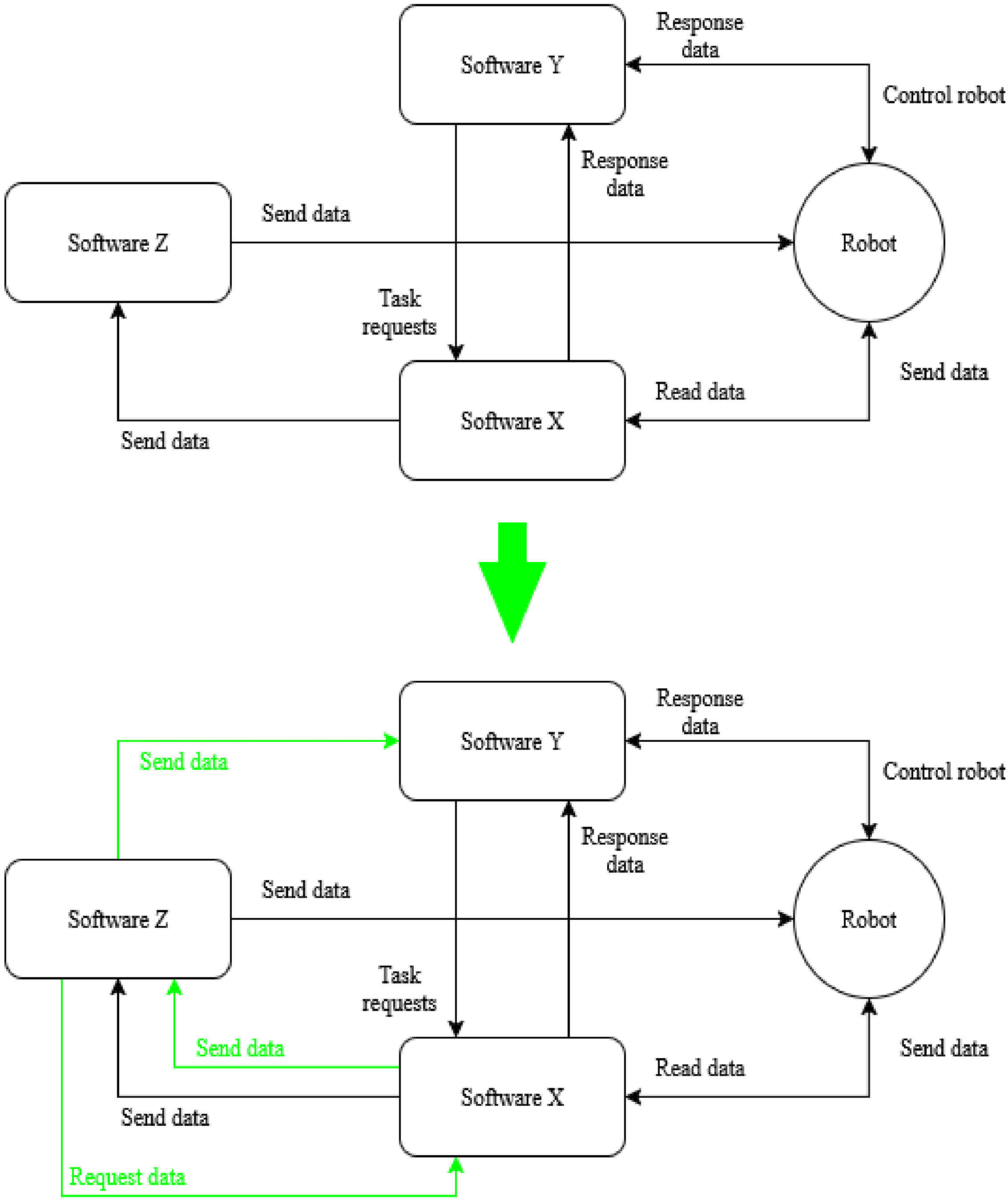

Diagram 3: Communication diagram that shows what the prototype added into the systems' communication

The diagram above shows how the communications between the three software solutions and the robot was increased for the prototype that was created. After the improvements into the communication, all three software solutions are connected to each other and the ability to share data has increased.

These practical developments directly reflect the theoretical expectations identified in the literature review. Previous research emphasized that increasing communication between modular software components improves maintainability, reduces redundancy, and enhances workflow efficiency [4], [5], [10], [16]. On top of that, studies focusing on UX highlighted that minimizing user switching between interfaces and maintaining visual consistency across solutions strengthens usability, learnability and user trust [17], [18], [23], [24].

By implementing these theoretical principles in the prototype, particularly the enhanced data exchange between software X, Y, and Z, this research was able to observe whether the expected benefits from the integration materialize in practice. Therefore, the prototype functions as an experimental validation of the literature. It tests whether integration improves user experience and whether that improvement leads to greater production efficiency in an industrial setting.

Next, we will discuss how the findings from the literature review and the prototyping process can be used to address the first research question.

4.4 Findings for Research Question 1

To understand how the findings of this research address the first research question, the focus is placed on the UX of the software system and how it was affected by the integration process.

The integration primarily targeted the communication between the three software solutions that form the complete system. By increasing this communication, both in the volume of data exchanged in previously existing connections and by introducing new connections between previously disconnected solutions, the system became more coherent and effective in its' use cases.

As discussed in the literature review, complex industrial systems often consist of several specialized software components, each with a distinct responsibility [14], [15], [19]. While this modularity supports maintainability and scalability, it often leads to fragmented workflows that negatively impact usability and operator efficiency [17], [18]. The findings from the prototype support these observations. Before integration, operators were required to interact with three software solutions to complete tasks, with one of them being a solution they lack expertise in, forcing them to switch between systems with different UI and contexts unnecessarily. This not only reduced efficiency but also increased cognitive load and the likelihood of human errors to occur, aligning with the challenges identified in previous studies on multi-system UX [17], [20].

The enhanced communication between the solutions enables the redistribution of user interaction points across the system. Tasks can now be performed in more appropriate contexts without requiring operators to use software components outside their intended scope and expertise. For example, prior to integration, operators interacted with Software X, which was originally designed for welding engineers with specialized domain knowledge. This mismatch increased the probability of human error and created inefficiencies. After integration, Software X is used by welding engineers for creating or updating weld patterns, while operators can complete their daily workflows without accessing it.

This redistribution clarifies the overall system structure, aligning the practical usage of each component with their intended responsibilities. Such alignment between user workflows and system design has been shown in prior UX research to improve learnability, reduce task complexity, and increase user satisfaction while reducing the amount of human errors occurring during the process [18], [22], [23]. Making the structure of the system clear for everyone to see from the way it is used, will add value to the marketability of the system also.

Furthermore, reducing the number of times users must switch between systems enhances UX consistency. Since each of the three software solutions features a distinct interface style, minimizing switching reduces visual and cognitive friction, as noted by studies focusing on UI consistency in industrial contexts [24], [25].

In addition, the use of gRPC connections provides a flexible technical foundation that simplifies future updates to data flow and interface interactions between the different software solutions. This adaptability reflects the literature's emphasis on iterative refinement and modular integration as enablers of sustainable UX improvement [5], [10]. This allows for the future improvement of the communication levels to be more efficient and for the future implementation of data gathering of the processes that the literature shows to have a positive impact on further making the production more efficient and secure [27].

In summary, the integration improved the UX of the system in two primary ways:

1. Reduced context switching – Even though the amount of user inputs required for the usage of the system cannot be nullified, users now have to perform less of those tasks and within fewer software environments, lowering cognitive load and improving efficiency.
2. Improved system clarity and role alignment – Each software solution's purpose and usage are now more intuitive and aligned with user expertise.

These findings suggest a significant positive improvement in UX as a result of the software integration, while also being consistent with theoretical expectations gathered from the literature review.

4.5 Findings for Research Question 2

The primary goal of the integration was to increase the automation level of the system that controls the automated welding station, thereby improving the overall efficiency of its operation. This improvement was achieved through enhanced communication between the software solutions and the relocation of certain points of user interaction, both of which reduced manual operator intervention and clarified system workflows.

As established in the literature review, increasing automation in industrial software systems tends to enhance production efficiency by reducing the time spent on manual data transfer, minimizing human error, and optimizing workflow coordination between subsystems [7], [10], [15], [21]. The prototype findings support these principles. The integration significantly decreased the amount of operator involvement in tasks that does not add value to the process by enabling software-to-software communication where operator input was previously required.

With the clarified structure to the system, fewer human resources are wasted on redundant or unnecessary operations that can be automated or even removed. Before the integration, operators were required to manage multiple interactions across different software solutions to execute routine tasks, and these tasks had a specific order that added complexity into the operator's workflow. After the integration, many of these actions are now handled automatically through direct communication between the software solutions and the order of the actions will be correct without the need for the operators to worry about them. This change aligns with previous findings that link automation and process synchronization with measurable improvements in operational efficiency which directly correlates with improved production efficiency [14], [16].

Furthermore, the clearer division of responsibilities between user roles enhances both production efficiency and safety, which will save wasted resources, helping with achieving environmental goals, and make the job of the operators more secure. Operators now focus solely on Software Y and Software Z, while Software X is used only by welding engineers for tasks that require further expertise regarding the knowledge of applying and modifying weld patterns. This separation has shown to reduce training time, lower the rate of operator errors, and allow users to concentrate on their core responsibilities [11], [13], [17]. As a result, operators can devote more attention to monitoring and managing the robot station, which indirectly contributes to increased productivity and safety.

The literature also emphasizes that improved UX and reduced task complexity directly correlate with production efficiency, particularly in automated manufacturing environments [18], [23]. The integration reinforces this connection. By streamlining workflows and minimizing cognitive load, users' complete operations faster and with greater accuracy.

In practical terms, the efficiency gains can be summarized as follows:

1. Reduction of manual steps – The complexity of the use cases of the system requires some user interactions, but some tasks previously performed by operators are now executed automatically and required data moves through the new gRPC communication channels from solution to solution.
2. Shorter learning curve and fewer errors – The system's clarified structure allows quicker onboarding and safer operation, both of which contribute to sustained efficiency over time and more efficient production.

While the prototype environment does not allow for a precise quantitative measurement of production efficiency after the designed integration, the observed improvements indicate a substantial qualitative increase in production efficiency. Based on the prototype's outcomes and alignment with established research on software-driven automation, it can be concluded that the integration leads to a notable improvement in production efficiency, primarily through automation, clearer workflows, role optimization and safer workflows that reduce possible errors and wasted resources.

4.6 Summary of Key Findings

The first research question asked; to what extent does UX improve with the software integration? The findings demonstrate that the increased communication between the three software solutions significantly improved the system's usability and overall UX. By enabling data to flow automatically between the three software solutions, several unnecessary tasks from the operators were removed. This reduced the need for operators to switch between the applications and eliminated redundant manual steps, both of which are known in the literature to enhance usability and consistency [17], [23], [24]. The relocation of user interaction points also meant that operators no longer needed to use Software X, which was originally designed specifically to be used only by the welding engineers, who have the required knowledge to use the solution effectively and safely. This change aligned the interaction structure of the system with users' actual roles and expertise, creating a more coherent, intuitive, and trustworthy UX.

The second research question asked; how much does production efficiency increase as a result of the integration? The results suggest a notable qualitative improvement in the production efficiency. The integration streamlined workflows by automating information exchange between the software solutions, reducing the number of manual operations required from the operators. This allowed operators to focus more on managing the welding station itself rather than on redundant software interactions. As supported by prior studies on automation and workflow optimization [7], [10], [15], [21], these changes directly contribute to faster task completion, reduced learning time, and fewer operational errors.

Overall, the findings indicate that the software integration successfully aligned the system's technical structure with its operator workflows. This alignment not only improved the user experience but also enhanced process efficiency by clarifying responsibilities, reducing redundant actions, and strengthening communication between the three software solutions. The integration therefore provides both usability and productivity benefits, offering clear evidence that continued development of the integrated design would be beneficial for the company's operational performance.

5 Conclusions

The software industry today faces a number of persistent challenges, including rising system complexity, quicker schedules for development, and the need to maintain high quality, security, and usability. To combat these challenges, software systems grow more distributed, and component based. With distributed systems ensuring reliability, usability, interoperability and maintainability have become increasingly difficult. These challenges are further amplified when industrial processes are automated, and software systems must interact seamlessly with physical systems and processes where safety is of top importance and the context of the use for the software is outside pure computational environments. Often industrial processes have already incorporated systems that have being built using already outdated technologies, which might have missing documentation and direct API calls might not be possible, further increasing the complexities regarding integration, automation and modernisation efforts. The constant pressure to enhance efficiency and usability, while managing integration across heterogeneous technologies, reflects the same issues that this research set out to address.

This research was conducted in collaboration with a company specializing in robotic welding automation stations. These stations are controlled by a software system composed of three distinct solutions. The system is divided in this manner due to the inherent complexity of the welding process and the way the robot is controlled. Without such separation, the UX of the system would suffer, affecting the efficiency of the use cases negatively, and making the development of the system too complicated for innovative features to be added into it.

Because certain aspects of the welding process cannot be fully automated, user interaction remains necessary. Prior to this research, operators of the welding stations were required to use all three software solutions on a daily basis. However, one of these solutions, Software X, was designed specifically for welding engineers with expertise in applying and modifying welding patterns. Its inclusion in the operators' workflow increased the likelihood of human error, as operators who lacked the correct expertise had the possibility to control aspects outside their area of responsibility.

The research addressed this issue by exploring the integration and improved communication between the separate software solutions. Through this integration, user interaction points were relocated so that Software X was no longer required in the operators' daily workflow. This not only enhanced the usability and UX of the system but also improved the efficiency of the welding process itself.

The research questions guiding this work were:

1. To what extent does UX improve with the software integration?
2. How much does production efficiency increase as a result of the integration?

The findings demonstrate that the integration significantly improved UX by reducing the need for users to switch between multiple solutions and by providing a clearer, more structured system. This reduced the learning curve and lowered the risk of human error. In terms of efficiency, the integration contributed in two ways: by clarifying the system structure, which improved safety and reduced error potential, and by minimizing manual interactions, thereby decreasing the time required for operators to complete their tasks.

This study was shaped by the company's specific needs, which imposed certain limitations. The scope was restricted to problems directly relevant to the company's context, and broader aspects were not investigated unless required by the industrial application. Nevertheless, the research successfully answered its stated questions and provided the company with valuable insights to guide decision-making about future system development.

In conclusion, this work contributed both practical and academic value, it demonstrated how software integration can improve UX and production efficiency in complex industrial systems, and it provided a concrete prototype to support implementation and future development.

Next steps regarding the prototype will include the further development of the features. Because the prototype was created by adding the barebones of the features, and by mocking one of the software solutions involved, the development process will start by adding all the required features into the software Y and by further developing the already existing features to be more concrete and then the fine tuning can start. After this, a first project, where the integration will be deployed, will be undertook. It will be used as a method for further testing the integration and all of its' features in an environment where nothing is mocked. Future work, when it comes to the topic of this research, could extend these findings by examining how similar integration approaches may be applied in other domains of industrial automation or in the same domain but in a different context of industrial manufacturing.

References

[1] Szalavetz A., 2020, “Chapter 3: Digital transformation and local manufacturing subsidiaries in central and eastern Europe: Changing prospects for upgrading?” in “The challenge of digital transformation in the automotive industry : jobs, upgrading and the prospects for development. Brussels : ETUI” , pp. 47-64, Available: http://hdl.handle.net/11159/4875

[2] Peldszus S., Brugali D., Struber D., Pelliccione P. & Berger T., 2025, “Software reconfiguration in robotics”, Available: https://link.springer.com/article/10.1007/s10664-024-10596-9

[3] Niswar M., Asfruddin R., A., Bustamin A. & Aswad I., 2024, “Performance evaluation of microservices communication with REST, GraphQL, and gRPC”, Available: https://ijet.ise.pw.edu.pl/index.php/ijet/article/view/10.24425-ijet.2024.149562

[4] El-Deeb A., 2022, “Major Challenges Currently Facing the Software Industry”, Availbale: https://dl.acm.org/doi/abs/10.1145/3539814.3539818

[5] Ihme T., Pikkarainen M., Teppola S., Kääriäinen J. & Biot O., 2014, “Challenges and industry practices for managing software variability in small and medium sized enterprises”, Available: https://link.springer.com/article/10.1007/s10664-013-9253-0

[6] Saeeda H., Ahmad. M. O. & Gustavsson T., 2023, “Challenges in Large-Scale Agile Software Development Projects”, Available: https://dl.acm.org/doi/pdf/10.1145/3555776.3577662

[7] Shen W., Hao Q., Mak H., Neelamkavil J., Xie H., Dickinson J., Thomas R, Pardasani A. & Xue H., “Systems integration and collaboration in architecture, engineering, construction, and facilities management: A review”, 2010, Available: https://www.sciencedirect.com/science/article/abs/pii/S1474034609000664

[8] Al-Saqqa S., Sawalha S. & Abdel-Nabi H., 2020, “Agile Software Development: Methodologies and Trends”, Available: https://core.ac.uk/download/pdf/482969304.pdf

[9] Terragni V., Vella A., Roop P. & Blincoe K., 2025, “The Future of AI-Driven Software Engineering”, Available: https://dl.acm.org/doi/abs/10.1145/3715003

[10] Crnkovic I., 2001, “Component-based software engineering – new challenges in software engineering”, Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/swf.45

[11] Rindell K., Ruohonen J., Holvitie J., Hyrynsalmi S. & Leppänen V., 2021, “Security in agile software development: A practitioner survey”, Available: https://www.sciencedirect.com/science/article/pii/S0950584920302305

[12] Tahaei M., Frik A. & Vaniea K., 2021, “Privacy Champions in Software Teams: Understanding Their Motivations, Strategies and Challenges”, Available: https://dl.acm.org/doi/abs/10.1145/3411764.3445768

[13] Ogunwole O., Onukwulu E. C., Joel M. O., Adaga E. M. & Ibeh A. I., 2023, “Modernizing Legacy Systems: A Scalability Approach to Next-Generation Data Architectures and Seamless Integration”, Available: https://www.allmultidisciplinaryjournal.com/uploads/archives/20250306182550_MGE-2025-2-018.1.pdf

[14] Jensen V. V., Alami A., Bruun A. R. & Persson J. S., 2025, “Managing expectations towards AI tools for software development: A multiple-case study”, Available: https://link.springer.com/article/10.1007/s10257-025-00704-7

[15] Ilyas M., Khan S.U. & Rashid N. 2020, "Empirical Validation of Software Integration Practices in Global Software Development", Available: https://www.proquest.com/docview/2932506309/fulltext/8633346B40874600PQ/1?accountid=27292&sourcetype=Scholarly%20Journals

[16] Land R. & Crnkovic I., 2003, “Software Systems Integration and Architectural Analysis – A Case Study”, Available: https://ieeexplore-ieee-org.ezproxy.cc.lut.fi/stamp/stamp.jsp?tp=&arnumber=1235441

[17] Saarijärvi J., 2017, “Improving User Experience through Consistency”, Available: https://aaltodoc.aalto.fi/items/c7bddb1e-fa96-4dd0-ae7a-9d1dd441099d

[18] Allam A. H., Hussin A. R. C. & Dahlan H. M., 2013, “User experience: Challenged and Opportunities”, Available: https://seminar.utmspace.edu.my/jisri/download/F1_FinalPublished/Pub4_UserExperienceChallenges.pdf

[19] Lallemand C., Gronier G. & Koenig V., 2015, "User experience: A concept without consensus? Exploring practitioners' perspective through an international survey.", Available: https://www.sciencedirect.com/science/article/abs/pii/S0747563214005718

[20] Sacolick I., 2022, "What is CI/CD? Continuous integration and continuous delivery explained", Available: https://www.proquest.com/docview/2650316466?accountid=27292&parentSessionId=8z0qx3RdDpnT83%2FExGqmWnol1MQZ04iHdoRTLOjieCo%3D&pq-origsite=primo&sourcetype=Trade%20Journals

[21] Torhnill A. & Borg M., 2022, "Code Red: The Business Impact of Code Quality – A Quantitative Study of 39 Proprietary Production Codebases", Available: https://arxiv.org/abs/2203.04374

[22] Badran O. & Al-Haddad S., 2018, "The Impact of Software User Uexperience on Customer Satisfaction", Available: https://www.researchgate.net/publication/329018487_The_impact_of_software_user_experience_on_customer_satisfaction

[23] Low Tze Hui S. & Lan See S., 2015, "Enhancing User Experience Through Customisation of UI Design", Available: https://www.sciencedirect.com/science/article/pii/S2351978915002383

[24] Ruissalo M., 2018, "Operating a design system in a large software company", Available: https://aaltodoc.aalto.fi/items/03a4c5ba-b6de-447c-ad3c-db32b0d69a30

[25] Ajiga D., Okeleke P. A., Folorunsho S. O., Ezeigweneme C., 2024, "The role of software automation in improving industrial operations and efficiency", Available: https://www.researchgate.net/profile/Daniel-Ajiga/publication/383410390_The_role_of_software_automation_in_improving_industrial_operations_and_efficiency/links/66cb6393c2eaa500231867e1/The-role-of-software-automation-in-improving-industrial-operations-and-efficiency.pdf

[26] Monostori L., 2014, "Cyber-physical production systems: Roots, expectations and R&D challenges", Available: https://www.sciencedirect.com/science/article/pii/S2212827114003497

[27] Wuest T., Weimer D., Irgens C. & Thoben K., 2016, "Machine learning in manufacturing: advantages, challenges, and applications", Available: https://www.tandfonline.com/doi/full/10.1080/21693277.2016.1192517

[28] Aldalur E., Suárez A., Curiel D., Veiga F. & Villanueva P., 2023, "Intelligent and Adaptive System for Welding Process Automation in T-Shaped Joints", Available: https://www.mdpi.com/2075-4701/13/9/1532

[29] Biber A., Sharma R. & Reisgen U., 2024, "Robotic welding system for adaptive process control in gas metal arc welding", Available: https://www.researchgate.net/publication/379361445_Robotic_welding_system_for_adaptive_process_control_in_gas_metal_arc_welding

[30] Bjarnason E., Lang F. & Mjöberg A., 2023, "An empirically based model of software prototyping: a mapping study and a multi-case study“, Available: https://link-springer-com.ezproxy.cc.lut.fi/article/10.1007/s10664-023-10331-w

[31] Rudd J., Stern K. & Isensee S., 1996, "Low vs. high-fidelity prototyping debate", Available: https://dl.acm.org/doi/abs/10.1145/223500.223514

Appendix 1

Diagram 1: Communication diagram of the current system

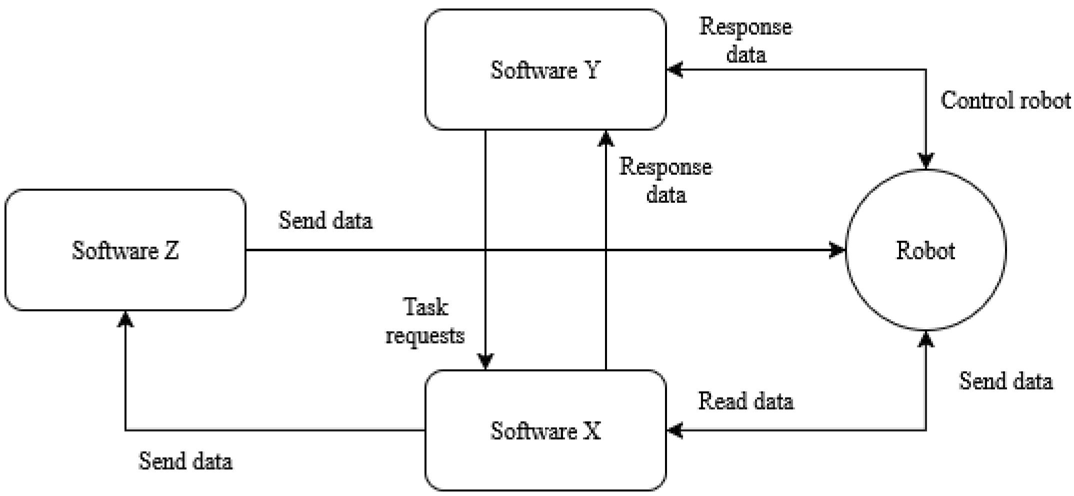

Diagram 2: Communication diagram of the prototype

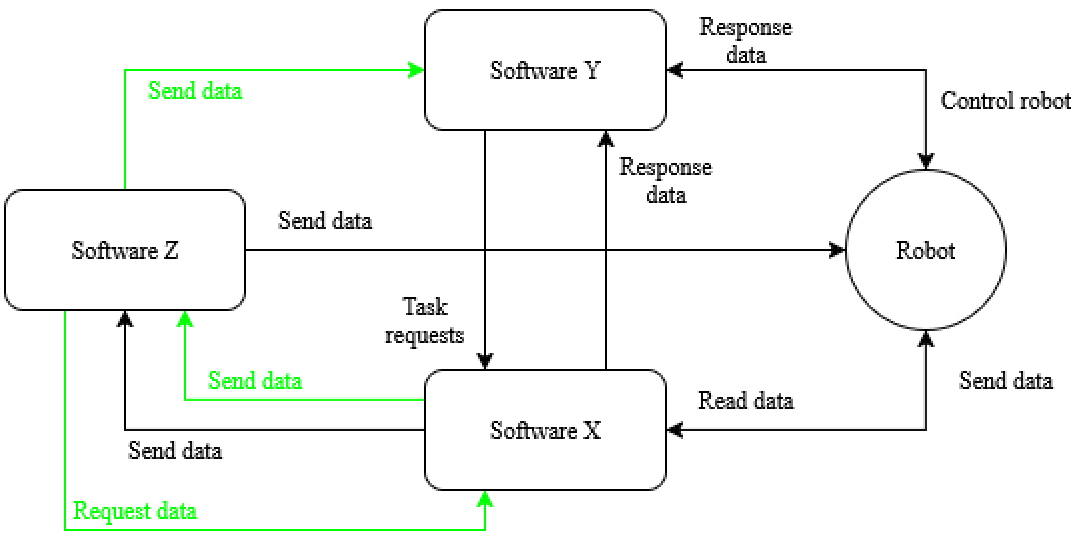

Diagram 3: Communication diagram that shows what the prototype added into the systems' communication